Automatic data transfer from CAD to CAPP for prismatic components

Linardakis, Socrates

Award date:
1996

Awarding institution:
University of Bath

AUTOMATIC DATA TRANSFER FROM CAD TO CAPP FOR PRISMATIC COMPONENTS

Submitted by Socrates Linardakis

for the degree of

Doctor of Philosophy

of the University of Bath

1996

COPYRIGHT

Socrates Linardakis

January 1996

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY

This research is aimed at automating the data transfer between the manufacturing functions of Computer Aided Design (CAD) and Computer Aided Process Planning (CAPP). It focuses on prismatic parts, typically produced in a small to medium batch manufacturing environment. The industry standard CAD DXF file format of a typical three view engineering drawing is used as the system input.

An interpreter module has been developed that processes the DXF file, filtering out any redundant data and keeping only the information necessary for manufacturing feature identification and process planning. The results are recorded in an intermediate file for further processing by the main system, BUFIP.

BUFIP (Bath University Feature Identifier for Prismatic components) forms the main core of the research work, and constitutes the developed manufacturing feature identification system. It is built using a modular structure (for expandability and ease of maintenance) and features a comprehensive error detection and reporting system. The system accepts for its input the interpreted file of a prismatic component. Various developed feature recognition algorithms are then applied from separate module programs, which are able to identify a range of manufacturing features, along with the associated characteristics necessary for process planning. Recognised features include flat features, such as multiple through slots, multiple stepped faces or their combinations, as well as three different types of pockets (open, side or closed). Cylindrical features are also covered, including plain holes, stepped holes (counterbored or countersunk) and threads.

Once the manufacturing features and their characteristics are identified, the system proceeds with dimensional tolerance allocation, currently implemented for the overall dimensions of a prismatic part. It also determines a component's block shape envelope (the minimum block of raw material required to machine the part) even when no dimensioning or tolerancing data are provided. The results of these processes are automatically recorded in an output data file using the ASCII format. This should make them directly usable by an appropriate CAPP system, or easily checked and edited by human operators if needed. The system has been tested and fine-tuned over a wide variety of diverse prismatic part drawings. It has been found to be robust, and able to resolve complex component profiles successfully within the research work's limitations.

TABLE OF CONTENTS

LIST OF FIGURES

Page

Page

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Dr. A.R. Mileham, who, being most generous with his time and providing many helpful suggestions, comments, ideas and criticism, made this research work possible. His valuable guidance is sincerely appreciated.

I would also like to thank the EPSRC, my colleagues, my friends, and all the people who have contributed directly to this work.

Above all I am deeply grateful to all my family and Irene, for their continuous support and encouragement through all these difficult years.

CHAPTER 1

INTRODUCTION

1.1 Computer Integrated Manufacturing (CIM)

The manufacturing industry of today is under considerable pressure to look for alternatives to the traditional approaches to design, manufacture and management. Faced with rapid change due to worldwide competition, advancing technology, and a greater tendency towards product variety (leading to a higher incidence of batch manufacturing), companies need to react to changes much faster, and in a more flexible way than in the past. They need to increase design productivity, create effective manufacturing systems, optimise the information flow, and make correct decisions in a very short time. To achieve this, they have adopted the extensive use of computers [Lau and Norrie 1996].

The application of computers within manufacturing organisations has been one of the major changes within the industry. Developments in hardware and software have made the computer an indispensable aid in solving complicated manufacturing problems and improving performance. Many manufacturing concerns have used computers to considerable advantage for many years in various areas of manufacturing, particularly design and drafting, machine control, production control and order processing [Kalpakjian 1995]. In all of these cases the computer has been used to perform a job previously done manually in the same way, but more

efficiently. However, the automation of separate functions does not guarantee a global optimum solution to the manufacturing problem [Chryssolouris 1992].

Manufacturing systems still typically consist of islands of automation. A lot of research and application has been aimed at integrating these islands in a way that leads to efficient and radical change. This move towards integration is now widely known as *Computer Integrated Manufacturing (CIM)* and has become perhaps the most vogue topic in manufacturing [Singh 1996, Vaquero 1995].

Computer Integrated Manufacturing is described by [Mitchell 1991] as the deliberate integration of the various computer automated systems required to design and manufacture a product. CIM can be considered as the logical organisation of the individual engineering, production and marketing / support functions into a computer integrated system. From this definition it is important to note that manufacturing includes not only technical activities, but also business functions such as marketing, finance etc. The technical activities include: Computer Aided Design (CAD), Computer Aided Process Planning (CAPP), Computer Aided Manufacturing (CAM), Computer Aided Quality Control (CAQC). These components of the integrated computer system and their relationship to the model of manufacturing are illustrated in **Figure 1.1**.

To gain maximum benefit out of computer integration, the decision maker should have access to all the data on all relevant computers, together with the use of computers to analyze the data [Udo and Udoka 1992, Weston 1995]. Much research and development work has been carried out worldwide to integrate these

manufacturing functions within CIM [Black 1996, Czajkiewicz and Wielicki 1994, Doumeingts et al. 1995]. The most notable attempts for integration are those between CAD and CAM.

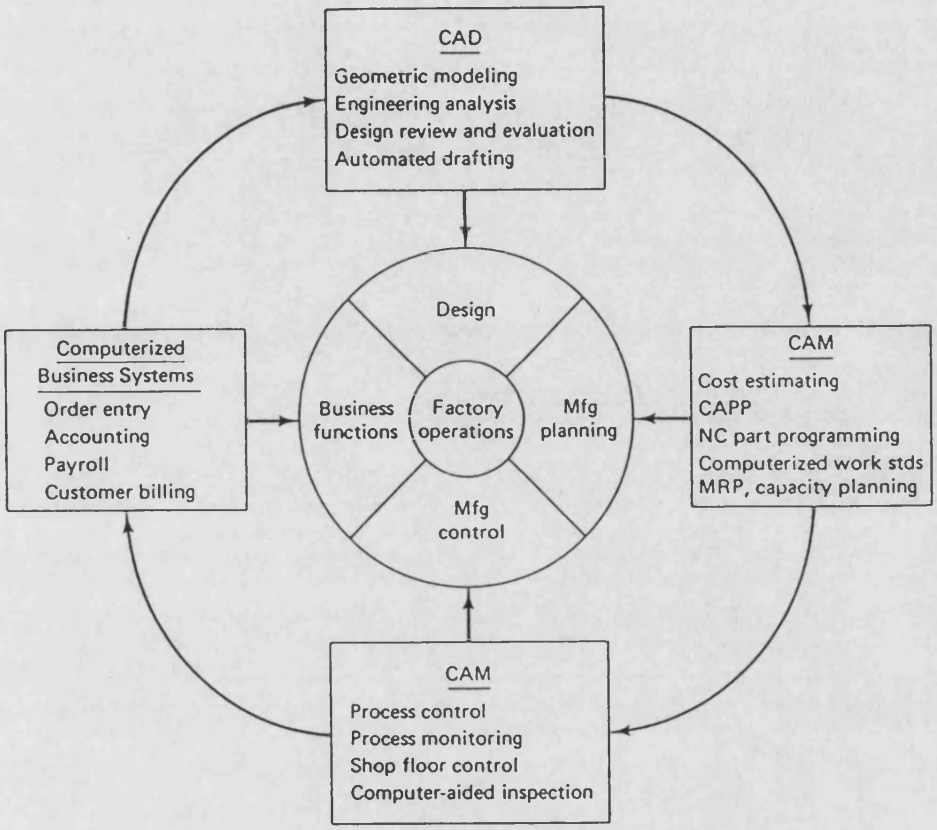


Figure 1.1 Computerised elements of a CIM system [by Groover 1987]

The CAD/CAM systems developed during the 1970s and early 1980s were designed primarily to address engineering problems [Groover 1987]. With CAD/CAM a direct link is established between product design and manufacturing engineering. The goal of CAD/CAM is not only to automate certain phases of design and certain phases of manufacturing, but also to automate the transition from design to manufacturing within CIM systems [Bedworth et al. 1991]. However, an

integrated CAD/CAM system cannot exist without an automated process planning system, linking the two functions by utilising design data from a CAD system and information from manufacturing databases to produce documents and/or machine instructions for the manufacture of components. Therefore CAPP is an important link between CAD and CAM, and a key factor in developing CIM systems [Rembold et al. 1993, Yura et al. 1994]. An overview of CAPP, as well as the CIM components it links (CAD and CAM) is presented in the following sections.

1.1.1 Computer Aided Design (CAD)

Computer Aided Design (CAD) can be defined as any design activity that involves the effective use of the computer to create, modify, or document an engineering design [Black 1996]. CAD is most commonly associated with the use of an interactive computer graphics system, referred to as a CAD system.

There are several important reasons for using a computer aided design system to support the engineering design function [Suh 1990]:

1. *To increase the productivity of the designer.* This is accomplished by helping the designer to conceptualize the product and its components. In turn this helps to reduce the time required by the designer to synthesize, analyze, and document the design.

2. *To improve the quality of the design.* The use of a CAD system with appropriate hardware and software capabilities permits the designer to do a more

complete engineering analysis and to consider a larger number and variety of design alternatives. The quality of the resulting design is thereby improved.

3. *To improve design documentation.* The graphical output of a CAD system results in better documentation of the design than what is practical with manual drafting. The engineering drawings are superior, and there is more standardisation among the drawings, fewer drafting errors, and greater legibility.

4. *To create a manufacturing database.* In the process of creating the documentation for the product design (geometric specification of the product, dimensions of the components, materials specifications, bill of materials, etc.), much of the required database to manufacture the product is also created.

The various design related tasks which are performed by a modern CAD system can be grouped into four distinct categories [Krueger 1995]:

(1) Geometric Modelling.

(2) Engineering Analysis.

(3) Design Review and Evaluation.

(4) Automated Drafting.

Since this research focuses on input to a process planning system, Geometric Modelling should be examined in more detail.

Geometric modelling is concerned with the use of a CAD system to develop a mathematical description of the geometry of an object [Bedworth et al. 1991]. The mathematical description, called a *model*, is contained in computer memory. This permits the user of the CAD system to display an image of the model on a graphics terminal and to perform certain operations on the model. These operations include

creating new geometric models from basic building blocks available in the system, moving the images around on the screen, zooming in on certain features of the image, and so on. These capabilities permit the designer to construct a model of a new product (or its components) or to modify an existing model.

There are various types of geometric models used in computer aided design. One classification distinguishes between two-dimensional and three-dimensional models [Jones 1992]. Two-dimensional models are best utilized for design problems in two dimensions, such as flat objects and layouts of buildings. Three-dimensional CAD systems are capable of modelling an object in three dimensions. The operations and transformations on the model are done by the system according to user instructions in three dimensions. This is helpful in the conceptualization of the object since the true three-dimensional model can be displayed in various views and from different angles. Geometric models in CAD can also be classified as being either wire-frame, surface or solid models.

A *wire-frame model* uses interconnecting lines to depict an object. Wire-frame models of complicated geometries can become somewhat confusing because all of the lines depicting the shape of the object are usually shown, even the lines representing the other side of the object. Techniques are available for removing these so-called "hidden" lines and fitting surfaces over the edges etc, which help resolve the potential ambiguities. Even with these improvements however, wire-frame models sometimes exhibit certain well-known limitations (such as ambiguity, non-completeness or non-uniqueness of the model compared with the object modelled). Despite these limitations, wire-frame models are still very widely used in industrial

CAD, particularly for 2D and 2 1/2D components [Zhao et al. 1993], mainly due to their simplicity (and hence low requirements for computer memory and processing time). Most commercially available CAD packages have been developed on wireframe modelling systems. Within these systems, the components are represented with lower level entities, such as lines, circles and arcs. In manufacturing, components are usually described using standard engineering drawing practice (BS308). This utilizes descriptions that are closely related to wireframe model entities and those are also the most widely used information media through the different stages of manufacturing (particularly NC machining). Therefore, wire-frame models still possess an important manufacturing significance.

Surface modelling overcomes many of the ambiguities associated with wire-frame models. It can provide a more detailed description of a part's surface geometry in that surfaces and boundaries are defined more precisely. Surface modelling is helpful in producing smooth continuous surfaces in NC machining and is mainly used in industry to design the curved surfaces of aircraft, ships or automobiles. However, the CAD database of a surface model does not have the intelligence to associate the surface into a single entity which encloses a volume [Woodwark 1986].

Solid models represent the most advanced form of geometric modelling. In this approach, an object is modelled in solid three dimensions, providing the user with a vision of the object very much like it would be seen in real life. More important for engineering purposes, the geometric model is stored in the CAD system as a three-dimensional solid model, thus providing a more accurate representation of the object. This is useful for calculating certain properties of the object, to perform

interference checking between mating components in an assembly, and in other engineering computations.

There are several internal representations used in solid modelling. Internal representation is the way a model is represented in the computer. It differs from the external representation which is what humans see on a computer monitor. External representation is generated from an internal representation by certain transformations. Solid modelling internal representations include Constructive Solid Geometry (CSG), Boundary representation (B-Rep), Sweeping, Spatial Occupancy Enumeration, Cell Decomposition and Primitive Instancing [Mantyla 1988]. Amongst these the most commonly used are CSG, B-Rep and Sweeping. These representations completely and unambiguously represent a solid model in the 3D space. Since the topology and geometry of the object are complete, they can be used by many engineering applications directly. However solid modelling places a heavy demand on computer memory and processing time.

Although a solid model represents an engineering object completely and unambiguously, it represents only the nominal dimension. There is still no standard way of modelling tolerances in a solid model [Krueger 1995]. With the exception of a few cases [Requicha and Chen 1986, Turner and Anderson 1988], tolerance information and annotations (something essential to manufacturing) are not part of the solid modeller. Applications of solid modelling are mainly for drafting, and attempts to include others such as NC cutter path generation have only been successful for models of a very restricted domain [Krigman 1992]. Despite its disadvantages, solid modelling will probably become the primary method for

geometric modelling in the future, especially with the continuous drop in the cost of computing processing power.

Finally, two other features in CAD system models are colour and animation. Some CAD systems have colour capability rather than only mono. The value of colour is largely to enhance the ability of the user to visualise the object on the graphics screen. For example, the various components of an assembly can be displayed in different colours, thereby permitting the parts to be more readily distinguished. Animation capability permits the operation of mechanisms and other moving objects to be displayed on the graphics monitor. Neither of these features are currently relevant to CAPP.

1.1.2 Computer Aided Manufacturing (CAM)

Computer Aided Manufacturing (CAM) can be defined as the effective use of computer technology in the planning, management and control of the manufacturing function [Puttre 1992]. The applications of CAM can be broadly divided into two categories [Trego 1995]:

1. Manufacturing planning.
2. Manufacturing control.

The two categories represent two different levels of involvement of the computer in the operations of the plant.

With manufacturing planning, the computer is used indirectly to support the

production function, but there is no direct connection between the computer and the process. In other words, the computer is used "off-line" to provide data for the effective planning and management of production activities. Important applications of CAM in this category include computer aided process planning (CAPP), cost estimating, computerised machinability data, computer assisted NC part programming, computer aided line balancing and production and inventory planning.

On the other hand, manufacturing control CAM applications are concerned with developing computer systems for implementing the manufacturing control function, often in real time. Manufacturing control deals with managing and controlling the physical operations in the factory. Such operations include process control, shop floor control and process monitoring, with the computer used "on-line" (directly connected to the process).

From the manufacturing point of view, CAD is a part design function and CAM is a part fabrication function. To integrate CAD and CAM there must be a link to connect them together. This link is provided by Computer Aided Process Planning (CAPP) [Fuh et al. 1995].

1.1.3 Computer Aided Process Planning (CAPP)

The task of process planning in manufacturing systems is to determine the sequence and procedure of the individual manufacturing processes required to transform raw material into a finished part [Kusiak 1990]. Traditionally the processes

and their sequence are documented on a form called a "procedure sheet", which details the results of various decisions required to manufacture the part, including:

- The selection and sequencing of processes.
- The selection of the machine tool set and the cutting tool set.
- The calculation of machining variables (speeds, feeds etc) and the identification of non-machining elements (handling times, quality checks etc).
- The selection of workpiece holding devices.
- Cost estimation.

Process planning is the most important interface between design and manufacturing, since many production functions such as capacity planning and scheduling etc, depend on process planning information. Despite the importance of process planning however, it is still perhaps the weakest link in today's manufacturing systems, due to it requiring significant amounts of both time and experience [Gu and Norrie 1996]. The complexity of the planning task is also further increased by the characteristics of modern manufacturing systems (small volumes, large variety).

In manual process planning, the process planning procedure is very much dependent on the experience and judgement of the planner. It is not unusual for different planners to specify different routes for the same part, each expressing their own preference. Furthermore, there is no easy way of ensuring that any route is optimal, and hence the level of planning efficiency will affect the efficiency of manufacturing. Manual process planning also reveals a variety of problems such as

the lack of expertise, inconsistency of the plans, and it suffers from both low productivity and a decreasing number of experienced process planners in industry.

The problems with manual planning, and the need to automate the process planning functions has led to the use of computers to assist in all process planning tasks. Computers offer the potential for reducing routine clerical work, while at the same time they are capable of calculating complicated formulae and analyzing logic rules in a much faster time. Process planning systems assisted by a computer are called *Computer Aided Process Planning (CAPP)* systems.

A change from manual process planning to Computer Aided Process Planning (CAPP) could provide for a batch working company benefits in the following areas [Mileham et al. 1988]:

- Increased planning productivity (up to 600%).
- Reduced lead times.
- Improved documentation, better consistency, legibility and less error.
- More consistent planning.
- Can provide the basis for a truly integrated CIM system if successfully interfaced with a CAD system, by facilitating the efficient transfer of information from CAD.

Much research work has been carried out to investigate the feasibility of using a computer to perform the process planning tasks. Three different approaches have been used for this purpose; they are discussed in detail in **Chapter 2**. Although many computer aided process planning systems have been developed, particularly in the

research stage, complete automation of the tasks involved has proved difficult to achieve due to the complexity of the problem [Kamrani et al. 1995].

1.2 Overview of the Research Problem

In today's rapidly changing and increasingly competitive manufacturing environment, the commissioning of an effective CIM system is seen as a necessary step to improve the operational efficiency, competitiveness and profitability of a company. Among the technologies employed within CIM, CAD/CAM is perhaps the most important topic, which requires that CAD and CAM be integrated into a total manufacturing system. Despite the successes of CAD and CAM in their individual domains, they still suffer from a problem similar to that in the traditional flow of information from design to manufacturing: lack of communication.

A database produced by a CAD system contains the information of a component in the form of purely geometric entities and text which can be called a *CAD product model*. The various processes comprising CAM however (and that includes CAPP, the link between CAD and CAM), normally require a database in a form of machining features, tolerances, and surface finish information, which can be called a *manufacturing product model*. In order for CAD to be integrated with CAPP by automatic data transfer, an interface must be built, able to convert a CAD product model into a manufacturing product model automatically. This is considered to be probably the most important and most difficult task in computer aided process planning [Chang 1990]. Most of the existing CAPP systems use people to act as the

interface, in which either a GT-like coding scheme or a part description language is used to translate design information into the process planning specific data (such as machining features).

Furthermore, the complexity of the problem is considerably increased if the CAD model describes a prismatic component. Prismatic components, with their complex 3D nature, their (at least) six plane surfaces and their usual lack of symmetry are notoriously difficult to automatically translate from CAD data to process plans. This is the reason that most commercial and research CAPP systems already presented [Eversheim and Schneewind 1993] deal mostly with the simpler, axi-symmetric rotational parts.

Very few CAPP systems incorporating an automatic interface between CAD and CAPP have been reported, particularly for prismatic components. They are examined in more detail in **Chapter 2**. So far, there has not been any universal system that can automatically transfer *both* the geometric and technological data directly from CAD to CAPP for prismatic components. Therefore, an effective CAD interface that can exchange all prismatic component data directly for use in CAPP systems is needed to enable comprehensive integration to take place.

This work presents the conception, design and development of such an interface for prismatic components, which has been designed to significantly add to the knowledge and techniques that are required to overcome many of the weaknesses currently present when converting CAD drawings to process plans automatically.

1.3 The Research Objectives and Aims

This research is concerned with automating the input of information into Computer Aided Process Planning (CAPP) systems for prismatic components. It concentrates on how information at the design stage in the form of an industry standard CAD data file can be interpreted and processed automatically into a manufacturing feature based output file, to be used directly by a CAPP system.

The primary aim of this work is to develop an automated CAD to CAPP interface for prismatic parts, typically produced in a small to medium batch manufacturing environment. The main objectives for the proposed system are:

- To enable automatic prismatic component data transfer directly from a CAD system, including dimensional tolerances and possibly surface roughness data.

- To maintain compatibility with a wide range of CAD systems currently used in industry.

- To incorporate algorithms for the automatic recognition of an extensive variety and combinations of prismatic parts surfaces and manufacturing features.

- To develop an internal classification and coding scheme for the workpiece and its features.

- To provide a methodology for selecting the most appropriate size of standard raw

material required to machine the component.

- To generate the final output documentation automatically, having all the information required for process planning and subsequent manufacturing.

- To provide the output data in a standard format, directly usable by a CAPP system, as well as maintaining the ease of checking and editing by human operators should the need arise.

- To indicate its potential for direct interfacing with a CAPP system by examining a particular application example: BEPPS-GSCAPP [Rustom 1992].

- To provide a means of easily updating and upgrading both the algorithms and the range of features the system can detect.

- To be easy and friendly to use even by an unskilled operator.

- To provide a cost-effective solution for small to medium batch manufacturing companies.

The developed system should be efficient in terms of computer memory and processing time, while at the same time being sufficiently robust in its operation and comprehensive in its error detection and reporting. This will be verified by testing and validating the system over a wide range of prismatic components.

1.4 Research System Limitations

Since the proposed system is to be used initially for research purposes, certain boundaries have been placed to limit its scope and logic. The main imposed limitations are:

- Only prismatic components with vertical and horizontal surfaces and features are considered. These are to be machined from solid blocks of raw material only.

- Certain features are not allowed to overlap (a more detailed explanation of these can be found in **Chapter 5** and **Chapter 6**).

- Only dimensional tolerances for a component are included in this study, applicable to the overall dimensions of the part, not its individual features. Geometric tolerances such as straightness, roundness, parallelism etc, as well as surface roughness values are excluded.

- Only the DXF industry standard CAD file format, as applied to PC compatible versions of CAD systems is considered.

- The consideration of workpiece holding devices is excluded from this study.

- Components with "protruding" features (features that extend out from the main block shape of the component, for example "islands") are not included in this study. This also applies to features that are "obliquely" cut (eg holes drilled at an angle from the horizontal or vertical axes).

CHAPTER 2

REVIEW OF LITERATURE

2.1 Introduction

Computer Aided Process Planning (CAPP), as the name suggests, uses a computer to determine the process plan for a part. A significant number of researchers and industrial groups have been focusing on this area since the 1970's. During this period, several CAPP systems have been developed using one of three approaches adopted for this purpose [Gu and Norrie 1995]:

1. The constructive approach.
2. The variant approach.
3. The generative approach.

Although in this research the interface between CAD and a generative CAPP system is developed in detail, the interface is considered to be generic to all CAPP systems. Therefore, for completeness, all of the above approaches are discussed in this chapter, along with a brief overview of typical CAPP systems exemplifying them. Feature Technology, an important element in process planning, is also examined. Various attempts at CAD-interfaced systems are also reviewed.

2.2 The Constructive Approach

This approach uses a computer database containing manufacturing information such as available materials, machines, cutting tools etc. Typically, the user constructs process plans interactively, with the computer providing alternatives and some recommendations to assist the user to make a decision. However, it still relies on the planner's expertise.

The process plan is formulated from a library of standard phrases. Once a machine type has been identified, the system presents a list of available machines to the planner to choose from. When this choice is made, appropriate speeds and feeds are automatically chosen, and the operation time calculated.

Constructive process planning has the benefits of flexibility and high data consistency. However, a human planner is still required to make the decisions. The constructive approach to process planning has not featured in published research for many years. Never the less, there are still several applications of constructive process planning systems in industry, which, despite their disadvantage of low automation, have been claimed to substantially increase planning productivity.

Typical constructive CAPP systems include AUTOCAP [El-Midany and Davies 1982] (a system developed in UMIST for turning components), C PLAN [CADCENTRE], LOCAM [PAFEC] and SOFIE 2 [OD Engineering Systems].

2.2.1 LOCAM

LOCAM was developed by PAFEC in Nottingham [PAFEC]. It makes use of a database which stores pre-determined manufacturing rules and time standards. Single items or tables, along with their description, are used to save standard information for individual operations. The planner selects appropriate elements from the standard data (the values of which depend on parameters stored by the program) and the system then combines them in the manner specified by the planner. The system also requires assistance to determine the sequence of operations (this has to be compiled by the planner).

2.3 The Variant Approach

The variant approach to process planning is comparable with the traditional manual approach where a process plan for a new part is created by recalling, identifying and retrieving an existing plan for a similar part (sometimes called a standard or family part) and making the necessary modifications for the new part. In some variant systems parts are grouped into a number of part families, characterised by similarities in manufacturing methods and is thus related to Group Technology (GT). For each part family, a standard process plan, which includes all possible operations for the family is stored in the system. Through classification and coding, a code is built up by answering a number of predefined questions. These codes are often used to identify the part family and the associated standard plan. The standard plan is retrieved and edited for the new part. This approach is particularly suitable

for component populations that can be readily grouped into families of similar shaped components. It can be used effectively and economically if there are few component families and many similar components in each family [Furth 1988].

Part families are formed by considering one of the following criteria [Gu 1991]:

(1) Design attributes (eg geometric shape, overall size).

(2) Manufacturing attributes (such as the sequence of processing steps required).

(3) A combination of design and manufacturing attributes.

For each part family, a composite standard process plan, which includes all possible operations for the family, is established by experienced process planners and stored in the system. Through classification and coding techniques based on Group Technology concepts, a code for each component and family is built up by answering a number of predefined questions. These codes are then used to organise the composite plans in the database and to search for the family (and the associated standard plans) to which the components belong. The standard plan can thus be easily retrieved and subsequently edited for the new part.

Compared with manually performed process planning, the variant approach offers certain advantages [Chang 1990]:

- It improves planning efficiency, particularly in the batch manufacturing industry where similar components are produced repetitively.
- It increases the information management capabilities. Consequently, complicated activities and decisions require less time and labour.

- Procedures can be standardised by incorporating a planner's manufacturing knowledge and structuring it to a company's specific needs. Therefore, variant systems can organise and store completed plans and manufacturing knowledge from which process plans can be quickly evaluated.
- Once a standard plan has been written, a variety of components can be planned with minimal expert input.
- A system can be easily extended by adding new family plans.

However, variant systems also have some disadvantages:

- The biggest drawback is that the quality of the process plan still depends on the knowledge background of the process planner. The computer is just a tool to assist the manual process planning activities.
- There are difficulties in maintaining consistency in editing practices, and an inability to adequately accommodate various combinations of geometry, size, precision, material, quality and shop loading.
- Inefficient plans are perpetuated.
- The effort that has to be put into coding and classification of the components in order to form the part families could be very large (depending on the complexity and quantity of the components).
- Systems have low flexibility due to their retrieval nature, since alternative standard plans should be prepared for different volumes of production (components in the same family may need quite different processes when the batch size increases).
- A variant system is of little value if there are many families and few similar components, ie high variety low volume systems.

Although research in variant process planning has virtually stopped in recent years, many variant process planning systems have been developed, and in fact, many of the existing process planning systems currently used in industry are based on the variant approach [Gu and Lorrie 1995]. Typical examples include CAM-I's CAPP, MIPLAN and MULTICAPP and ICAPP.

2.3.1 CAM-I CAPP

CAPP is an acronym for "CAM-I's Automated Process Planning system" developed by Mc Donnell Douglas Automation Company (McAuto) under a contract from CAM-I [Link 1976]. It is probably the first and also the most widely used of all process planning systems. CAPP is a database management system written in ANSI standard FORTRAN. It was developed primarily as a research tool to demonstrate the feasibility of computer assisted process planning, with logic based on group technology methods to classify and code parts.

A structure is provided for a database, retrieval logic, and interactive editing capability. The coding scheme for part classification and output format are added by the user. A 36-digit maximum alphanumeric code is allowed. A coded scheme tailored to the user application is usually appropriate.

2.3.2 MIPLAN and MULTICAPP

Both MIPLAN [Schaffer 1980] and MULTICAPP were developed in conjunction with OIR (Organisation for Industrial Research Inc.). They are both variant systems that use the MICLASS coding system for part description. They are data retrieval systems which retrieve process plans based on part code, part number, family matrix, and code range.

By inputting a part code, parts with a similar code (user-defined similarity) are retrieved. The process plan for each part is then displayed and edited by the user. They are similar to the CAM-I CAPP system with MICLASS embedded as part of the system.

2.3.3 ICAPP

ICAPP ("Interactive Computer Aided Process Planning") is a variant process planning system for prismatic parts written in FORTRAN on a VAX 11/750 under the VMS operating system [Eskicioglu and Davies 1983]. The system is feature oriented and capable of processing plane and cylindrical types of features. It is intended to be used for parts produced on conventional drilling, boring and milling machines as well as machining centres.

A composite part for ICAPP is designed for each part family and parts in each family can be derived from its composite part. The variant planning data and

the parameters of the generative logic of each composite part are kept in the Cutting Technology File (CTF). The system then selects the necessary machining operations and cutting parameters for each part from the CTF, according to the part's feature type, dimensions and tolerances. The process planning sheet produced by ICAPP can be modified manually if required.

2.4 The Generative Approach

With the generative approach, process plans are created using decision logic, formulae, technological algorithms and geometry based data to perform uniquely the many processing decisions for converting a part from raw material to its finished state. Generative process planning can thus be defined as a computer based system that synthesises process information in order to create a process plan for a new component automatically [Shah et al. 1991].

Thus, instead of storing standard process plans as in the variant systems, generative systems should be capable of generating process plans based on the information provided by the part drawing, relevant engineering specifications, and information regarding manufacturing resources available. To accomplish this, generative systems incorporate their own knowledge base, consisting of a manufacturing database and decision logic that imitates the human planner. The manufacturing database includes the part description data and the technological information such as machining and tooling data [Houtzeel 1995].

A generative process planning system consists of three main components [Chang 1990]:

(1) Part description.

(2) Decision making logic and algorithms.

(3) Manufacturing database.

2.4.1 Part description

The *part description* contains all the geometrical and technological data required to generate the process plan for the part. For generative systems, input of the part description can come either as a text input where the user answers a number of questions in an English or English-like dialogue (defined as interactive input) or as graphics input where the part data is gathered from a CAD module (defined as interface input). So far the former is more common in existing CAPP systems, while the latter is still a fairly undeveloped area due to its complexity. Interactive input can take the form of a code description or as a descriptive language.

Code description techniques are typically based on Group Technology [Kusiak 1987]. Parts are grouped into families based on feature or process characteristics. Each part in a family has the same code, despite differences in shape. The code number can define the part features and can also identify a component's design specification and its manufacturing attributes.

The advantage of this method is that the code is easy to generate and

manipulate because it is concise. However, coding is typically a manual process, and exact shape and size information, necessary for detail process planning, is lost when the part is described by a finite digit code. Automatic generation of GT codes which cover all the design specifications has not yet been effectively achieved [Ali and Motavalli 1993].

Therefore code based part description is not considered suitable for a completely automated process planning system, since it does not provide sufficient detail, and requires a human interface between the design and process planning functions. APPAS [Wysk 1977] is a typical example of a well known generative process planning system using coding schemes.

Descriptive languages on the other hand are special languages introduced to describe parts directly for the process planning function. Their format provides a means of describing the component in detail both geometrically and technologically. Many generative process planning systems, particularly those using expert system approaches, employ a special part description language.

Descriptive languages appear in many different forms, but are usually easy to understand and formulate for simple components. For a complex component however, the translation from the part drawing to the input format can be very tedious and difficult. Another disadvantage of this approach is that the input description must still be carried out manually. GARI [Descotte and Latombe 1981] and AUTAP - NC [Eversheim 1982] are examples of systems using the descriptive language approach.

As far as interface input is concerned (component data obtained directly from a CAD system), it is examined in more detail in section 2.7.

2.4.2 Decision making logic and algorithms

The decision making logic and algorithms form the most important part of a generative CAPP system. The logic imitates the expertise used by the human process planner to make decisions on all aspects of planning, for example machine tool and machining parameter selection, cutting tools choice and operation sequencing. In this respect programs need to computerise judgement type decisions currently taken by people, as well as manage an increased complexity of such decisions, with much more available data requiring a quicker response [Sormaz and Khoshneris 1995].

To achieve these tasks, the decision logic has to be synthesised and structured in a clear, compact form with potential for easy updating and revising of the logic and rules. Methods developed to organise decision logic fall broadly into three main categories:

(1) *Decision Trees*: These are graphical representations of the decision logic, and represent a natural way to depict process information. The tree consists of a single root with branches emanating from it [Chang 1990]. Each branch corresponds to a specific condition, and is connected to other branches by means of nodes. The nodes provide further branching. Branches terminate in an action if the conditions specified to transverse the branch are true. Decision trees can be easily implemented as

computer codes, but, once developed, system expansion and updating can be very difficult.

(2) *Decision Tables*: Decision tables, as the name suggests, contain decisions and actions in a tabular form. They are usually preferred to decision trees because they are more easily expandable [Gu and Norrie 1995]. They are also easy to implement on a computer, and possess simpler maintenance and modification requirements. Both these approaches however cannot cover all the process planning decision logic required (for example they cannot represent operation sequencing). This has led to the development of Artificial Intelligence (AI) techniques.

(3) *Artificial Intelligence Techniques*: Artificial Intelligence (AI) has become an important tool in the process planning field. Successful development of AI techniques has led to the use of expert systems in process planning [Gupta 1990, Kiritsis 1995]. The structure of an expert system closely resembles the intelligence of the human expert. Expert systems have the capability of using domain knowledge intelligently to suggest alternative courses of action [Singh 1996]. They consist of two major components:

(a) The Knowledge Base representing the expert knowledge (rules and facts).

(b) The Inference Engine representing the control mechanism for using the knowledge base. This can be one (or both) of two main types [Rembold et al. 1993]; Forward Chaining (FC) or Backward Chaining (BC).

Many expert planning systems have been developed to solve the process selection and planning problems [Sabourin and Villeneuve 1994, 1996, Roy et al.

1995, Zhang 1993, Rozenfield et al. 1992, Zust and Taiber 1990]. Such systems can possess considerable advantages over the other two approaches, for example the separation of the knowledge rules from the other parts of the system (hence allowing for their easy revision), the ability to improve their own knowledge based on experience and the ability to fully justify and explain any decision they make. However, their successful implementation still depends on the nature of the problem [Kiritsis 1995].

2.4.3 Manufacturing database

Databases of part description libraries, material types and specifications, tooling etc form the final part of a generative process planning system. They consist of information organised into a number of fixed-format records with logical links between associated records. They are used to provide the process planning system with the information required to make various decisions, and normally contain company-specific information. They should be designed in a way that facilitates easy updating of the files or changes of the system configuration.

2.5 Generative CAPP Systems

CAPP systems have been under development for more than two decades. Numerous generative CAPP systems have been reported, particularly at the research stage. The following sections provide a brief overview of some of the more important

generative CAPP systems that have been put forward. More detailed listings can be found in [Ajmal and Zhang 1994, ElMaraghy et al. 1993, Eversheim and Schneewind 1993, Alting and Zhang 1989, Ham and Lu 1988, Requicha and Vanderbrande 1988].

2.5.1 APPAS, CADCAM and TIPPS

APPAS is an acronym for "Automated Process Planning and Selection" developed by Wysk and described in his dissertation at Purdue University [Wysk 1977]. It is probably the first well known generative CAPP system and is written in standard FORTRAN with description of the detailed technological information of each machined surface being input by means of a special code. An implicit decision tree approach is used to directly code the process capabilities in the system. APPAS is capable of selecting multiple passes and processes for the designed machined surface (such as twist drill, rough bore, finish bore). Multiple diameter holes with special features such as an oil groove, slot or thread can also be planned. APPAS includes the selection of feed rate, cutting speed, diameter of the tool, number of milling cutter teeth, length of the tool and length or depth of cut for each tool pass. CADCAM is an extension of APPAS developed by Chang and Wysk [Chang and Wysk 1985], which links APPAS with an interactive computer graphics terminal to demonstrate the concept of an integrated CAD and process planning system.

TIPPS is an acronym for "Totally Integrated Process Planning System" developed by Chang and Wysk at Virginia Polytechnic Institute and State University

in 1982 [Chang and Wysk 1985]. In a sense TIPPS is a new generation of APPAS and CADCAM. It is probably the first system that attempts to integrate CAD and generative process planning into a unified system utilizing the AI and decision tree approaches. A special language called PKI (Process Knowledge Information) is used to describe the procedural knowledge (process capabilities). A CAD boundary representation as data input is used by the system. The user applies the crosshair cursor on a graphics terminal and menu display to the surfaces to be machined in order for the system to then determine manufacturing processes, sequence, cutting parameters and time estimates.

2.5.2 AUTAP and AUTAP-NC

AUTAP is an acronym for "Automatisch Arbeits Plannerstellung" developed at the Laboratory of Machine Tools and Production Engineering (WZL) at Aachen Technical University in Germany [Eversheim 1982]. It is one of the most complete generative process planning systems in use today. AUTAP-NC is similar to AUTAP. The only difference is that AUTAP is for the generation of process plans and AUTAP-NC is for the generation of part programs. Depending on the requirements of companies, the systems can be applied in an interactive or batch mode. AUTAP and AUTAP-NC are parts of an integrated system for the generation of manufacturing documents that has been used successfully in various German companies. AUTAP works as follows: (1) determination of the raw material; (2) determination of the operation sequence; (3) selection of the machine tools for each operation; (4) calculation of the estimated time; (5) determination of the operation

instructions. The AUTAP-NC works as follows: (1) determination of manufacturing segments; (2) determination of tools and cutting data; (3) selection of lathe chucks; (4) determination of the manufacturing sequence; (5) generation of the part program. The system can handle different kinds of rotational parts like shafts, discs, rings, gear wheels, bearing caps, as well as sheet metal parts which are components for telecommunication equipment.

2.5.3 BEPPS-NC

BEPPS-NC is a generative process planning system for rotational parts developed at Bath University [Zhang 1991, Zhang and Mileham 1991, Zhang and Mileham 1989]. It uses a 2D wire frame product model as input, typical of various CAD systems in the format of DXF (Drawing Interchange File). The system consists of four major modules:

1. *The CAD Interpreter.* This is able to extract the useful engineering drawing entities from a DXF file format, identify machining features from these entities, and rebuild the part product model in a feature based manner. Tolerance information can be automatically extracted and allocated to the relevant features, but surface roughness data have to be typed in interactively.

2. *The Process Planner.* This performs all the process planning tasks, such as set-up methods, operation selection and sequencing, cutting tool selection, machining parameter determination etc. After planning, a hard copy of the process plan is

printed.

3. *The NC Code Generator.* This is able to determine the cutting toolpaths and generate a NC program for a given part based on the process plan generated by the Process Planner.

4. *The BEPPS-NC Viewer.* This is aimed at helping the user to check the validity of component data and inspect toolpaths. It produces an image of the component and performs an interactive graphical simulation of the machining process.

2.5.4 BEPPS-GSCAPP

BEPPS-GSCAPP is a generative process planning system for prismatic parts developed at Bath University [Rustom 1992, Rustom and Mileham 1992, Rustom and Mileham 1990, Rustom and Mileham 1989]. It is aimed at parts being produced on conventional machine tools in a batch manufacturing environment. The system is of particular interest since it was chosen as an application example to indicate this research work's potential for direct interfacing with a CAPP package, hence it is described in more extensive detail, particularly its input stage.

The general structure of the BEPPS-GSCAPP software consists of four main options:

(1) User's help.

(2) Process planning.

(3) Decision logic modification.

(4) Database file modification.

The user's help option provides general guidance on how to use the system initially. In options three and four, the user can have access to both decision logic files and database files to enable updating and/or modification whenever it is required. A specially formatted file is designed to compile the system files automatically whenever any modification takes place. However, the main option of the system is process planning, which is divided into three stages: interactive (input) stage, automatic (planning) stage and output stage.

At the interactive-input stage, the planner types into the system the data required in order to generate the process plan. This stage is sub-divided into five distinct sections:

(1) General Information Data Input

This contains general information about the component, including its name, material, shape envelope (length, width and depth), batch type and size etc. Once this information is completed, a header file for the process plan sheet is automatically saved.

(2) Component Classification and Coding

After the general information regarding the component has been typed in, it is classed by the system into one of three main categories (flat, long or cubic). The system then awaits the planner (who must be familiar with it) to code, using

appropriate schemes, both the planes and edges that form the block shape envelope in which the component lies.

The planes have to be coded in such a way as to enable the planner to subsequently input the machining features in a distinct order for each plane of the component. For this purpose, six surface planes are defined, 3 datum and 3 opposite. They are named with reference to the axis to which they are normal to (ie X, Y or Z) and coded respectively. Hence the 3 datum planes are termed and coded as XD, YD, ZD while the 3 opposite planes as XO, YO and ZO. It should be noted that the longest dimension must always lie on the x-axis and the shortest dimension on the z-axis. The planner, who initially has to conceive any component as a block and then assign the codes for the plane surfaces, must be familiar with the plane codes to input them correctly whenever requested.

The edges are coded in an anti-clockwise direction according to their plane positions in order to recognise the location of a feature and for determining the machining direction. Hence the original x-axis edge is coded as X0 with subsequent edges coded as EX1, EX2 and EX3. In the same way, y-axis edges are coded EY0, EY1, EY2 and EY3, while z-axis ones as EZ0, EZ1, EZ2 and EZ3.

(3) Component Type

The entire component is further classified according to its shape specification, ie the flat manufacturing features required to be machined on it. This classification relies on the profile of the features across a plane surface, and their machining direction.

Three main component types are defined:

(i) Totally Constant Cross-Section if each of the surfaces that require machining have a constant profile in any plane direction.

(ii) Partially Constant Cross-Section if any one surface of those requiring machining has a constant profile in any one plane direction (more than one surface must require machining).

(iii) Non Constant Cross-Section if none of the surfaces requiring machining has a constant profile in any one plane.

The planner is requested to identify the type of component during data input, bearing in mind certain restrictive conditions that must also be satisfied.

(4) Feature Data Input

The system considers a range of machined features commonly produced on conventional machines for prismatic parts. The simplified research version uses 8 features namely flat surfaces, slots, steps, pockets, plain holes, stepped holes, countersunk holes and threads (not fully implemented). The eight feature types have been classified into two major groups, flat (including faces, slots, steps and pockets) and cylindrical (the various holes and threads). Each group is then subdivided into Basic and Secondary features.

The feature data for each plane surface that requires machining is put into the system interactively via system prompts. For each feature, the planner is asked for a variety of characteristics including location, dimensions, tolerances, surface roughness requirements etc. These data are stored in a component database file,

which includes the possibility of accepting feature information directly from a future CAD interface package.

(5) Machine Tool Availability

A machine tool database has been defined in BEPPS-GSCAPP, limited for research purposes to seven machine tools: a vertical and a horizontal milling machine, a pillar and a radial drill, a vertical boring machine, a surface grinder and an internal grinding machine. The system displays the machine tools (names and codes), so that the planner is able to delete machines that are currently occupied.

Once the input of data has been completed, the system stores the information in a file and proceeds to the automatic (planning) stage. Process planning is divided into 8 modules as following:

1. Raw material selection from stock.
2. Feature ordering.
3. Operation determination and sequencing.
4. Machine tool selection.
5. Cutting tool selection.
6. Cutting conditions selection.
7. Total time calculation.
8. Workpiece holding device consideration.

When the process plan has been completed in full, it is printed in the form of a planning sheet that contains the following elements:

1. Header, including general component and production information.

2. Process plan, divided into three sections according to the component design and machinability. These are cut-to-length, rough-and-finish flat and rough-and-finish cylindrical. The system stores the generated process plan initially as a temporary file; the planner then has to decide whether to keep it as a permanent file for future retrieval or not. If it is to be saved however, the planner cannot add, delete or modify any of its data in its present form.

2.5.5 CMPP

CMPP is an acronym for "Computer Managed Process Planning" developed by United Technologies Research Center (UTRC) in conjunction with the US Army Missile Command [Dunn and Mann 1978]. It was reported to be one of the most technologically advanced systems in use at the time, written in FORTRAN 77 and running on both Univac and IBM systems. CMPP which is the base for CAM-I's XPS-I and XPS-II [Sack 1983], is a generative process planning system aimed at high technology, machined cylindrical parts; parts characterised by expensive materials, tight tolerances and complex manufacturing processes. CMPP addresses requirements such as: (a) turning, grinding and honing to produce cylindrical surfaces; (b) milling, electrical discharge machining for other non-cylindrical features; (c) operations such as plating and heat treatment that apply to both cylindrical and non-cylindrical features. CMPP works in four steps:

1. Generating a summary of operations.
2. Selecting dimensioning reference surfaces for each cut in each operation.
3. Determining and analyzing machining dimensions, tolerances and stock removal

for each surface cut in each operation.

4. Generating process plan output.

An English-like problem oriented language COmputer Process PLanning (COPPL) is used to define manufacturing practice for parts. CMPP is interfaced to many CAD and CAM systems in American aircraft companies.

2.5.6 EXCAP

EXCAP is an acronym for "EXpert Computer Aided Process Planning" developed by Davies and Darbyshire [Davies and Darbyshire 1984, Joseph and Davies 1990] at UMIST. It is an expert system for generating process plans for the machining of rotational components. Actually EXCAP was developed from AUTOCAP, an initial CAPP system from UMIST written in BASIC. Three generations of EXCAP prototypes have been developed. They are EXCAP-A, EXCAP-Y and the more up to date EXCAP-P. The implementation language has been switched from BASIC to PASCAL and PROLOG running on a SUN 3 under the UNIX operating system.

The system uses a discrimination net for sequencing operations which is a set of decision trees connected together to form a network. The route taken through this network during planning has operations associated with it, and these operations form the manufacturing sequence. The system uses considerable amounts of heuristic knowledge to constrain the search space in order to increase efficiency.

EXCAP utilises a backwards planning strategy for operation selection and sequencing. It identifies the area to be machined first and selects appropriate filling-in operations in steps towards the blank. Tests of the system have been carried out using real components and company-specific knowledge obtained from industry.

2.5.7 TURBO-CAPP

TURBO-CAPP is one of the most complex intelligent generative process planning systems to date developed by Wang and Wysk [Wang and Wysk 1987] at the Pennsylvania State University. The system consists of five modules: (1) machine surface identification; (2) process selection and sequence; (3) NC code generation; (4) knowledge acquisition, and (5) database management. Each module is composed of several routines undertaking a specific task. It works in the following steps:

(a) Interpretation of geometrical data from a 2 1/2D CAD system.

(b) Extracts surface features from the CAD database.

(c) Checks design consistency of geometric dimensioning and tolerancing.

(d) Manages an extensive manufacturing knowledge base.

(e) Updates the knowledge base by interacting with experienced planners.

(f) Performs intelligent reasoning based on extracted machined surfaces and their relationship with surface finish, geometrical dimensions and tolerances, current machine configurations, and available tools in order to generate process plans.

A combination of both frame and rule-based methodology is used to build up the knowledge base of the system. In brief, process planning knowledge is structured

in three different layers: layers of facts, layers of inference rules, and layer of meta-knowledge for ease of reasoning. The system is implemented on PCs in Prolog.

2.5.8 XCUT

XCUT is a system focused on operation planning for prismatic parts on multi-axis CNC milling machines [Brooks et al. 1987]. Its unique feature is its knowledge base structure. This is represented by four different schemes: rules representing heuristic knowledge; definitive objects represent conceptual objects; relational tables represent domain knowledge and data; and lisp functions represent procedural knowledge.

Planning is started with the user identifying features through the use of an interface. After all features are identified, they are passed to the expert planning system. This includes feature planning, cut planning, cutting tool planning, cut plan optimisation and cut plan detailing. The rule bases in XCUT can be treated much like subroutines in a conventional language. At the time of reporting, the system contained approximately 500 rules and could only plan simple features such as pockets, notches, slots, steps and holes.

Further reported generative CAPP systems include:

AUTOPLAN [Vogel and Adlard 1981].

CCSPLAN [Zhao and Baines 1992, 1992, 1993].

CUTTECH [Barkovy and Zdeblik 1984].

GARI [Descotte and Latombe 1981].

GENPLAN [Tulkoff 1981].

KAPLAN [Giusti and Santochi 1989].

KAPPS [Iwata and Fukuda 1987].

OMEGA [Sabourin and Villeneuve 1994, 1996].

PART [Van Houten et al. 1989].

PC-CAPP [Pande and Walkevar 1989].

PRICAPP [Pande and Walkevar 1990].

SIPP and SIPS [Nau and Chang 1985].

XPLAN and XPLAN-R [Lenau and Alting 1986].

XPLANE [Van't Erve and Kals 1986].

XPS-1 and XPS [Sack 1983].

As it can be seen from the previous reportings, the major development thrust in CAPP systems occurred during the 1980s. In more recent years, research interest in this area has diminished considerably, with very few systems being reported. Instead, researchers seem to have focused their efforts on optimizing specific aspects of CAPP systems, or experimenting with modern concepts applied to the more general issue of automated process planning. These include genetic algorithms [Awadh et al. 1995], object-oriented methods [Gu et al. 1994], neural networks [Mei et al. 1995, Huang and Zhang 1995], and fuzzy decision-making logic [Zhao 1995]. Hybrid approaches have also been mentioned [Duerr et al. 1994, Zhang et al. 1994] which use a combination of established methodologies to achieve results. Attempts at optimization of specific CAPP sub-systems include operation sequencing [Taiber 1994, Gu and Zhang 1993], manufacturing capability modelling [Gao and Huang 1996] and workpiece modelling [Jovanoski and Muthsam 1995].

2.6 An Overview of Feature Technology

According to many researchers, the use of feature technology is the key to the genuine integration of CAD and CAM as part of a total Computer Integrated Manufacturing environment. Features are derived from the reasoning processes used in various design, analysis and manufacturing activities [Cunningham and Dixon 1988] and are usually strongly associated with particular application domains. Hence there are a number of definitions of features, according to the domain they originate from.

Pratt and Wilson [Pratt and Wilson 1985] broadly define features in the engineering domain as

"A feature is a region of interest on the surface of a part".

Henderson [Henderson et al. 1990] relates a feature's definition to the representation and recognition methodology:

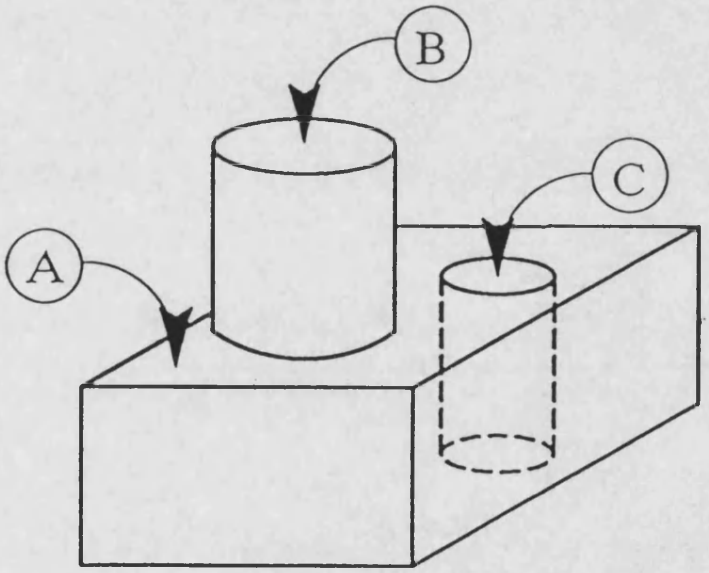
"Features are defined as geometric and topological patterns of interest in a part model and which represent high level entities useful in part analysis".

These definitions are generalised enough to cover all reasonable possibilities. However, by specialising, one could see the same features from at least three points of view: they can be seen as *design features, manufacturing features* or purely *geometric features*. **Figure 2.1** (after [Case and Gao 1993]) graphically illustrates this point.

When the geometry of a feature is being considered, the feature is then

termed a *geometric feature* (or "form" feature [Lenau and Mu 1991]). *Design features* are features considered by designers in terms of their geometry, specifications and details to fulfil certain functional requirements (thus they are sometimes referred to as "functional" features). Such features include, for example, keyways, cooling slots, and fitting holes.

The same features however, may be viewed differently by process planners or production engineers as *manufacturing features*. For example, a cooling slot could be seen as a general slot milled by a milling machine; a fixing hole can be considered as a drilled or bored hole etc. In general, design features are expressed in geometric terms, while manufacturing features express explicitly the methods of production, while implying the geometry and function of the features.



Geometric:	Manufacturing:	Design:
A. Plane Face	A. Machined Region	A. Mating Surface
B. Cylinder (positive)	B. 'Island'	B. Boss
C. Cylinder (negative)	C. Through Hole	C. Fixing Hole

Figure 2.1 Different interpretations of the same geometry (after [Case and Gao 1993])

2.6.1 Feature taxonomies

Instead of specifying all the geometrical and topological information that defines a feature for every separate feature type, it is possible to group features with common properties into classes. These can then be further divided into sub-classes to form a tree structure, or hierarchy. These classification structures are commonly called *feature taxonomies* and since they are of a hierarchical nature, the properties of a class can be inherited by its sub-classes.

The use of taxonomies is an important factor when developing a feature-based system. They allow large amounts of varied features to be classified into coherent groupings. This helps in the recall of previously defined features, their subsequent editing, and the design of new features. The hierarchical description of features also allows simple features to be combined into more complicated features, and hence the final model [Case and Acar 1989].

Another advantage of feature taxonomies is that they can provide a framework for the parametric generation of geometry at the design stage. Also, without a rigorous taxonomy, it is difficult to produce analytic and predictable algorithms for the complex task of process planning [Requicha and Vanderbrande 1989]. Several different taxonomies have been developed by researchers.

Gindy [Gindy et al. 1993,1992,1991,1989] uses a taxonomy which consists of form features at its top levels: depressions, protrusions and surfaces. Each of these is then sub-divided according to the number of external access directions present

(these are the directions from which the feature can be machined by cutting tools). Feature geometry is described by defining the external access directions, boundary type (open or closed) and the exit boundary status (through or blind). The taxonomy has a limited domain, since it does not consider rotational or sheet metal components. However it is adequate for classifying features used in a number of process planning applications.

Pratt and Wilson [Pratt and Wilson 1985] developed their taxonomy for CAM-I. It is similar to Gindy's since it also describes form features. Features are divided into two main sections: explicit (where all the geometric parameters of the feature are fully defined) and implicit (where sufficient information is supplied to define the feature, but the full details have to be calculated when required). The implicit method has the advantage of a smaller amount of data, whereas the explicit method is easier to analyze. Both forms can then be split further into compound or simple. Simple explicit features are then classified into through holes, protrusions, depressions, and areas with possible sub-division in terms of their cross-sectional shapes as rotational or prismatic. This provides a wider range then Gindy's taxonomy, with higher flexibility.

Butterfield et al. [Butterfield et al. 1985] also produced a taxonomy for CAM-I. They classified form features into three main categories: sheet, prismatic and rotational. Sheet features are sub-divided into flat or formed (with further sub-divisions), prismatic features into depressions, protrusions and surfaces (again with further sub-divisions) and rotational features into concentric or non-concentric. Other manufacturing information such as materials and heat treatments is also categorised,

providing this taxonomy with an advantage over the other two.

The collected information on a part represented in terms of its features is known as the *feature model* of the part. There can be different feature models for the same part, depending on the different definitions of features. For example, there can be a design feature model, manufacturing feature model or geometric feature model. There are two main approaches used to create such feature models: Feature Recognition and Design-by-Features.

2.6.2 Feature recognition

Feature recognition can be defined as the process of identification of features within a geometric model of a component after its creation [Mantyla et al. 1996]. This allows the manufacturing of parts drawn using conventional CAD systems, which not necessarily use solid modellers. A feature recogniser then identifies and extracts features from the geometric model, and stores them in a separate database forming the feature model, from which the required machining operations to produce the part can be derived.

Most research work on feature recognisers has focused on extracting features from solid models. Very few research efforts have been reported on surface or wireframe models (despite their widespread use in industry), perhaps because of the difficulty involved in extracting features from them due to their potential complexity and ambiguity [Subrahmanyam and Wozny 1995]. These attempts are described in

more detail in section 2.7. As a result, while the numerous feature recognition systems based on solid modellers can be classified into broad categories according to their characteristics [Wang 1992], there is not such a classification scheme for surface or wireframe model recognisers. The most notable categories of solid model feature identifiers, along with typical systems exemplifying them, are highlighted below.

Systems based on *syntactic pattern recognition* seem to be a popular choice among researchers. Syntactic pattern recognition is analogous to syntax analysis in natural languages. A pattern is represented as a collection of primitives just as a sentence is composed of a group of words. In pattern recognition, complex patterns are represented by simpler sub-patterns and explicit relations among sub-patterns. The sub-patterns can be decomposed again into even simpler patterns. This decomposition process continues until the simplest patterns are derived, which are called pattern primitives [Brown et al. 1995]. To be easily recognised, these pattern primitives should be simple.

Once the pattern primitives are determined, the next step is to define rules that describe relations among these pattern primitives, so that valid composition of the primitives are guaranteed. These rules are called production or rewrite rules and are specified by a phrase grammar. The grammatical rules correspond to the final patterns to be recognised [Gu and Norrie 1995]. This is done through a parser that classifies the input patterns to a class. This process is very similar to natural and formal language processing, in which a sentence can be analyzed to see if it is grammatically correct. Syntactic pattern recognition is most suitable for two

dimensional pattern recognition [Joshi and Chang 1990].

An early example of a notable system utilizing the syntactic pattern recognition approach is that of Kyprianou [Kyprianou 1980]. Kyprianou used the boundary representation of a part to recognize the depressions (such as holes) and protrusions (such as boss). A rule-based system then used the results to produce a Group Technology classification code. Staley et al. [Staley et al. 1983] developed pattern grammars for different holes (represented by cross-sections). Given a string representing a hole using picture primitives, a parser determines which hole class the hole belongs to. Other examples of syntactic pattern recognition systems include the systems developed by Choi [Choi et al. 1984], and Jared [Jared 1991].

A similar approach to syntactic pattern recognition is based on *state transition diagrams and automata* [Iwata et al. 1992, 1986]. Using this method, part geometry is described by sweeping operations, and/or the union of swept volumes. The generating surface is described by ordered pattern primitives, together with technological information. Features are recognized using a state transition diagram where, instead of using grammars and primitives, the relationship between adjacent primitives is used. Automata on the other hand, can be seen as a composite grammar for an entire part family. This assumes the prior development of grammars for individual family members. Any parts falling into this family satisfy the automata. A system using the automata method is that of Milacic [Milacic 1985].

Another popular strategy for identifying features involves *decomposition* approaches. One of these approaches partitions a design model into several smaller

volumes (*volume decomposition*). To be usable the decomposed smaller volumes need to be manufacturing or design features in order that they can be used for process planning. A recognition step is needed after the decomposition step to find the semantics of the features [Shen and Shah 1994]. In general, the total material volume to be removed by machining can be found by a Boolean difference between the stock and finished part. This total volume must then be decomposed into units that correspond to practical machining operations that match machining features.

Examples of volume decomposition systems that utilize the convex hull approach (also referred to as the alternating sum of volumes) include that of Woo [Tang and Woo 1991A, 1991B]. Woo's method computed the difference between an object and its convex hull recursively until a null set was obtained (ie until the object equals its convex hull). The object could then be represented as a sequence of convex volumes with alternating signs. This type of decomposition was not always useful, as it could result in a removal volume that did not correspond to a single machining operation (an odd-shaped feature). Also, the base stock was sometimes of awkward shape, because it was the convex hull of the initial shape rather than standard bar stock.

Another well known approach to volume decomposition is that developed by General Dynamics for CAM-I [CAM-I 1985], employing the delta volume decomposition method. This consists of decomposing a part into sub-volumes, typically corresponding to machining volumes (delta volumes). Examples of delta volumes are slabs, pockets, notches etc. If the sub-volumes do not correspond directly to delta volumes, they can be further broken down into sub-delta volumes

which are matched with a specific machining technique. Feature intersections however can give rise to volumes which are difficult to interpret geometrically. Also, the problem of dividing a given volume into sub-volumes for an application is not completely solved.

A variant of the decomposition approach is *cell decomposition*, that is used for cutter path generation. With this technique the component is divided into cells, formed by a lattice of planes that are parallel to the major axes [Shah 1991]. The size of each cell is related to the cutter diameter, and the depth of cut. The height of the cell is equal to the depth of cut. The length of the cell is equal to the diameter of the tool. Starting from any location, adjacent cells are collective, with a direction defined for cell collection. The direction is the cutter path. Cell decomposition techniques are well-suited for generating roughing cuts. This is because the part is discretized into cells of a definite shape (usually cuboidal), which results in tool paths that are approximate representations of the boundaries.

Armstrong [Armstrong et al. 1984] used the cell decomposition strategy to produce a milled part for a given part and stock geometry. He classified each resultant cell as either a stock cell (implying that the entire stock in it should be removed), a part cell (non-removable material) and a semi-part cell (containing both part and stock material) with special algorithms designed to navigate through the cells. Setup and cutting tool information for roughing and finishing cuts could be derived by testing required paths against cells in a given order. Technological considerations like surface finish and tolerance information, or component fixturing were not covered by this method.

Yuen [Yuen et al. 1987] used a similar method to Amstrong, but employed octree subdivision for spatial ordering. Octree representations for the product and the workpiece are created, which are then converted into a quadtree form. The cutter path is generated by traversing on the plane of the quadtrees until a check surface is encountered. The tool traverses to the bottom of the part, until all the quadtree planes are covered. However, improperly finished profiles can result due to the differences between the actual and projected quadtrees of each layer.

Finally, the third variant of decomposition approaches is *decomposition by slicing laminae*, also known as *sectioning techniques*. With this method the part volume is sliced into horizontal laminae of constant cross-sections, in a direction perpendicular to the tool [Subrahmanyam and Wozny 1995]. The intersection of each plane and the part model (usually a B-Rep model) defines the boundary of the part at the plane. The NC tool path is generated by applying special algorithms to each of the horizontal laminae. Though the method works best for pockets with vertical walls, attempts have been made to machine general curved surfaces. Sectioning techniques are simple and similar to the method of offsetting tool path curves with respect to a part. However, the tool paths they yield are not optimal, and in the cases of non-planar or angular surfaces, major modifications are needed.

Examples of sectioning technique systems include Parkinson [Parkinson 1985], who used sectioning to deal with 3D faces (planes not parallel or perpendicular to spindle sections or a general curved surface), which were intersected with planes parallel to the xy or yz planes. The intersection curves thus produced were split into straight-line segments in order to keep within a given tolerance. These

segments were offset by the tool radius in the direction of the surface normal at their start and end points, and at a convex edge with a boundary face. Bobrow [Bobrow 1985] used a CSG solid modeller for input. His system also generated ordered parametric curves on the surface to be machined by slicing with parallel laminae.

Some solid model feature recognisers employ the *entity growing* approach. With this strategy recognised features are removed by adding / subtracting a volumetric shape that corresponds to the feature [Bronsvoort and Jansen 1993]. Since this feature might not always be a closed volume, new faces could be added to close the feature volume. This is known as entity growing, and can take the form of either face extension or edge extension. The disadvantage of this approach is that only convex volumes can be created.

Examples of systems using entity growing include Falcidieno's [Falcidieno and Giannini 1987], which extended edges or faces to generate volumes. It also created new edges and vertices. Sakurai and Gossard [Sakurai and Gossard 1988] generated feature volumes by adding half spaces corresponding to feature faces. The negative of the volume was then extracted from the object to continue feature recognition.

Another category of solid model feature recognisers uses *logical inference* (expert system) approaches. These are based on the logic used by a human in detecting features by attempting to capture the notion of a feature into some form of rules or logic [Wierda 1991]. Rules are usually represented in the form of IF-THEN statements and are created for each feature that needs to be recognised. The part

model is then processed according to these rules, using logical inference as the computational mechanism. Systems utilizing this approach are usually implemented in logical programming languages or production systems that directly support logical inference, such as Prolog or OP5. Since such systems however exhaustively search through all facts to prove each rule is true, they can be inefficient and computationally expensive. Also, although it is easy to develop rules for independent features, it is difficult to develop rules for intersecting features.

Pande and Prabhu [Pande and Prabhu 1990] describe such an expert system for identifying machined surfaces on symmetrical rotational components. They used the OP5PLUS expert system shell to represent the procedural knowledge to reason and extract the internal and external part features and their dimensions in order to select form tools for machining. All rules are expressed as "condition-action" pairs.

Vandenbrande and Requicha [Vandenbrande and Requicha 1990] developed their expert system methodology specifically with the intent of handling feature interactions. Hence instead of using rules to define features, they used rules to define feature hints, which are characteristic components of features. The logic behind this approach is that, since feature hints contain fewer topological entities than full features, they will be more likely to handle feature interactions without being modified. Feature hints are passed through several sets of rules which evaluate their potential and combine them into feature hypotheses. These hypotheses are verified using additional rules that implement various kinds of machining domain knowledge. The method is thus able to cope with relatively complex feature interactions.

Graph based approaches also seem to be a popular choice among researchers. With this method, features are modelled as stereotypical *subgraphs*, wherein the nodes and arcs represent all of the features' necessary topological components (usually faces and edges), and are labelled with all of the features' geometrical constraints. The B-graph is directly searched for subgraphs that match the feature graphs [Parry-Barwick and Bowyer 1993]. This approach category borrows a wealth of concepts and algorithms from various fields of mathematics, especially graph theory and topology. Most systems using graph based approaches extract features from the face-edge-vortex graphs of the B-Rep model (or their sub-graphs), since these contain sufficient information to describe an object. For example, Chuang and Henderson [Chuang and Henderson 1990] extract features from the vertex-edge graph; Henderson et al. [Henderson et al. 1990] extract features from the face-edge graph; and Van Houten et al. [Van Houten et al. 1989] extract features from the complete graph.

Joshi and Chang [Joshi and Chang 1988] use a graph based approach to process an Attributed Adjacency Graph (AAG), which is a B-graph in which the arcs are labelled with their edges' concavity. The separated subgraphs are searched for depression features only in descending order of feature complexity. Features are removed from the B-graph as soon as they are recognised, hence nested features can be handled. More complex feature interactions can result in some of the separated subgraphs having too many or too few arcs to be successfully matched. Joshi implements heuristic rules that split arcs and nodes (edges and faces) to form complete feature subgraphs, so at least some of these interactions can be handled as well.

A more recent example of a graph based approach is reported by Narayan and Ling [Narayan and Ling 1994]. Their system uses a set of heuristic rules to construct subgraphs from the graph representation of a design. The process of subgraph to feature identification is carried out with a set of integers and characters which represent the geometric, topological, and semantic characteristics of the corresponding feature.

Another general feature recognition methodology adopted by researchers is *rule based template matching*. A template usually is a Prolog predicate which consists of relationships satisfying a particular pattern to be matched. Each template corresponds to a feature (but sometimes for efficiency reasons it could represent a composition of several features). Rules are then used to look for certain patterns and relationships of elements in a part model, until some set of elements can be matched with the template and identified as a feature [Subrahmanyam and Wozny 1995]. This approach is suitable for knowledge based expert process planning systems.

Henderson [Henderson 1984] devised a rule based system for template matching by using concepts from both graph matching and syntactic pattern recognition. Features were formalised by templates that consisted of pattern rules. Templates were defined for both general features (such as holes) and specific features (such as flat bottomed, constant diameter holes). Rules were expressed as a set of both geometric and topologic conditions, each of which had to be tested separately; all conditions had to be satisfied for the rule to be satisfied. The recognition and extraction algorithm involved the following steps: determine cavity volume (difference between stock and part), recognise general features in each cavity,

classify general features into specific features, create and subtract the volume corresponding to each feature from the cavity, and repeat all the previous steps until there are no cavities left. This system was limited to sweep features only however.

Mortensen and Belnap [Mortensen and Belnap 1989] also proposed a system employing rule based template matching. Parts were described in a symbolic form, and the rules, embedded in predefined Prolog feature templates, searched to match certain sets of elements. Once a match was found, the corresponding feature was extracted.

The vast majority of the approaches discussed so far use Boundary Representation (B-Rep) solid models as input for the feature recognition process. Extracting features from a Constructive Solid Geometry (CSG) solid model is much more difficult because of the non-uniqueness of CSG binary trees, and hence has received very sporadic attention. Depending on the part designer, the same part model could have completely different CSG trees, requiring an almost unlimited number of templates or shape grammars to match them [Shah 1991]. To overcome this, a general approach adopted by some researchers involves converting the CSG tree into a *DSG (Destructive Solid Geometry)* tree, and then use the feature recogniser on the DSG model [Li and Yu 1990].

The DSG technique follows the manufacturing process of a 2 1/2D component closely by selecting a blank that the finished model can be contained within, and then deducting ("destructing") features in the order that they would need to be machined in practice, until the finished component is produced [Cutkosky et al. 1988]. Other

researchers re-construct the random CSG tree to from a unique and computer understandable tree, which is then identified [Herbert et al. 1990].

A typical example of the DSG approach is the system proposed by Perng et al. [Perng et al. 1990]. This converts the CSG tree into a DSG tree in which all primitives are disjoint and all nodes represent difference operations. The initial CSG tree is assumed to contain no intersection nodes, but it is straightforward to pre-process the tree to insure this. Every DSG primitive represents a removed volume which corresponds to some machining operation (except for the one primitive that represents the starting stock). Adjacent DSG primitives are combined to form composite volumes, and these volumes are classified as machining features according to the number and orientation of the faces by which they connect to the stock. The conversion process from CSG to DSG is fairly computationally expensive, in cases performing an exhaustive subdivision of the CSG primitives into disjoint pieces.

Lee and Fu [Lee and Fu 1987] devised a method consisting of two steps, namely feature extraction and unification. Since primitives used in CSG can each have an associated local coordinate frame, the principal axis is used in the case of feature extraction. Primitives such as cones, cylinders and tori can all be described by a single axis. A sphere can also be characterised using an axis with arbitrary orientation. Cubes on the other hand must use 12 axes because of their axis asymmetry. The approach uses the CSG tree as an input to generate all principal axes. The axes are then partitioned into several clusters based on spatial relationships. Within each cluster, the axes involved in a particular feature can be located according to the conditions defined by the feature. The feature representation is then unified by

rebuilding the CSG tree.

All the methods described previously use feature recognition on CAD solid models that have already been preconstructed. However, some researchers advocate the use of *hybrid approaches*, that combine feature recognition with the design-by-features strategy (described in more detail in section 2.6.3). It is argued that such a combination of both methods could potentially resolve the intractability of feature recognition and the inflexibility of design-by-features [Wang 1992].

Pratt for example [Pratt 1993] proposed that a designer could indicate to create a certain feature. The system would then prompt for the appropriate dimensional information and then generate a feature of the required type with the given dimensions. The feature is initially a separate entity until the designer indicates the required position and orientation on the main part model. The system would then position the feature and join it with the overall model. This approach would negate the need for model decomposition, and allow each feature to be validated for manufacturability etc before being attached to the model. However, the product still needs to be built using relatively inflexible feature primitives.

Laako and Mantyla [Laako and Mantyla 1993] have also implemented a hybrid feature modelling system. With their strategy, the designer has the flexibility to use either of the two approaches while designing the product model. The system is based on a feature identifier that provides "incremental" feature recognition; this allows changes to a geometric model to be recognised as new or modified features while preserving previously recognised features that remain unchanged in the

geometric model.

De Martino et al. [De Martino et al. 1994] propose a hybrid approach based on an intermediate model, which is shape feature based and provides a communication link between design by features and feature recognition. The method gives the designer the possibility of creating the product feature-based description using both features and geometry primitives which are subsequently used to generate the feature based model. The system also provides the choice of creating application specific feature taxonomies to map feature based descriptions between different application contexts. Conversion mechanisms transform a geometric model into the intermediate model, and from this to a context dependant feature based model, and vice versa.

The approaches described so far, adopted by reseachers for identifying features from solid models, cannot be used in detail for feature recognition from wireframe modellers. However, some of the developed principles could be utilised for feature extraction from wireframe models, as discussed in section 2.7.

2.6.3 Design-by-features

The design by features approach involves special design systems utilizing features for their internal component description, and hence eliminates the need for feature recognition altogether. With this approach, a geometric model is constructed using feature primitives from a library. Primitives such as slots, pockets, holes etc

or (more usually) blocks, cylinders etc, are used with operations such as add, delete and modify to create a feature representation of the component [Gao and Case 1992]. This representation maintains additional information not usually kept in a conventional solid modeller, such as feature names, taxonomy rules and attributes. However, systems at present have a finite feature library, and the feature operations such as add, delete etc. are frequently limited. Thus, they generally do not possess the design flexibility, versatility, application independence and wide industry acceptance that conventional CAD systems enjoy [Case and Gao 1993].

Design by feature systems do not necessarily incorporate a geometric modeller for their underlying data structures, but can be implemented in other ways, for example using languages that store the feature information in list format [Kramer 1988]. Another system not associated with a solid modeller is that of Dixon [Dixon 1988]. He proposes a knowledge-based design system consisting of two parts: the first consists of a user interface, features library, operations library and a monitor, allowing the user to create primary representations of features. The second part is used to convert the primary representations to secondary representations for further reasoning.

Abdala and Ikonopisov [Abdala and Ikonopisov 1993] also propose a knowledge-based design system used in conjunction with the Pro/Engineer solid modelling package. This commercial feature-based design package uses the method of Destructive Solid Geometry (DSG) explained before.

Pratt and Wilson [Pratt and Wilson 1985] also examined the representation

of features in a geometric solid model. They propose a feature processor that manipulates feature data and communicates with the modeller using a feature model database. The user interacts with both the modeller and the feature processor through an Applications Interface Specification. Shah [Shah and Rogers 1990] proposes a system consisting of a feature based modelling shell (integrated with a CSG solid modeller) and a feature mapping shell. The modelling shell incorporates all the facilities required for creating a component database, while the mapping shell extracts and reformulates part data for specific applications, hence enabling a wider integration with different applications. More recently, object-oriented techniques have been also used to enhance feature based design systems [Rahman et al. 1994].

Several other design-by-features approaches have also been reported by researchers. More detailed listings can be found in various papers [Case and Gao 1993, Lenau and Mu 1991].

2.7 Interfacing CAD and CAPP

From the description of some turnkey CAD/CAM systems available on the market today, it would appear that an "automatic" link between CAD and CAM already exists. Indeed, all CAD/CAM systems can generate some manufacturing information from an internally stored path description, although few can do more than specify individual machine toolpaths for a limited set of geometric surfaces. In all cases, human intervention is still required for process planning.

Therefore, one of the important challenges towards achieving the goal of true integration of CAD and CAM, is interfacing the CAD database with the CAPP module using a general neutral format without any human intervention. This would enable automatic product definition data input to the planning system, thereby eliminating the human requirement for translating the design into code or other descriptive formats. Since the 70's researchers have attempted to form a direct link between CAD and CAPP by using special formats to convert the CAD output data into a neutral format file to be used directly as process planning input data. However, because of the complexity of the task involved, few CAPP systems incorporating an automatic CAD interface have been reported.

The following sections highlight some of the attempts at automating the CAD/CAPP interface, that are of particular interest to this research work. This is because the following systems utilize wireframe CAD models for input (as does this work) and / or deal with prismatic components.

2.7.1 Meeran's system

S. Meeran [Meeran and Pratt 1993], [Meeran et al. 1993] proposed a method for recognising manufacturing features from 2D engineering drawings of prismatic components with orthogonal surfaces. The system uses the industry standard DXF CAD file format for input, and is implemented in Prolog. The overall process consists of five phases, namely:

(i) The post-processing of the DXF file representing the drawing into a sequence of Prolog facts.

(ii) The separation of the entities (lines, circles, arcs etc.) composing the drawing into three orthogonal views.

(iii) The determination of connectivities and hence closed loops of edges in each of the three views.

(iv) The correlation of patterns in the three views in accordance with rules for various types of commonly occurring simple machining features.

(v) The automatic generation of NC code to drive a machine tool for the manufacture of the part.

The actual feature recognition process (phase iv) is divided into three subphases:

(a) The extraction of simple isolated features.

(b) The extraction of slightly interacting simple features.

(c) The general classification of any remaining unidentified features (features not conforming to any of the feature rules implemented) as depressions or protrusions.

The system can identify a limited number of simple features from engineering drawings drawn using the third angle projection. However it only deals with features located on one side of a component only, and cannot handle tolerancing information.

2.7.2 Sakal and Chow's system

Sakal and Chow [Sakal and Chow 1994] developed a system to interface two affordable commercial PC-based CAD and CAM software packages: Autocad and Mastercam. The program, generative in nature, uses geometrical information from the internal Autocad drawing database to identify machinable features on the top and bottom faces of 2 1/2D orthogonal prismatic parts. It then automatically determines the Mastercam machining operations needed to machine the part. The system is implemented in Autolisp (Autocad's programming language) and modifies Autocad's menu system to provide the new choices for extracting features from a component.

Geometry extraction and organisation begins with a 3D wireframe representation of the prismatic part drawn using Autocad. The part may be composed of a combination of slots, steps, pockets or islands in either the top or bottom surfaces. The input geometry is limited to 2 1/2D. The actual package consists of many separate Autolisp subprograms which are loaded into Autocad as the functions are needed. Each of the subprograms may be classified into one of two categories: geometry extraction and feature recognition.

Artificial intelligence techniques are used to recognise the different machinable features directly from the Autocad database. As a first step all vertical lines from the wireframe model are removed. All of the line and arc segments comprising the part entities list are then stored according to their z coordinates. The result is a number of separate sublists, one for each z coordinate. Individual plane contours are then extracted from the sublists. In order to extract features from the set

of contours, two contour relationships are established, "matching" and "inside". Matching contours are those contours with the same x and y coordinates at different z coordinates, while "inside" contours are contours included within other contours.

Subsequent feature recognition consists of two steps: the determination of feature primitives and the combination of these primitives into a "feature tree". Feature primitives are determined using information from the "matching" and "inside" lists. These are searched for sets of contours satisfying predefined feature primitive criteria. Whenever a set is found, the names of all contours in the set are saved under the appropriate feature primitive. The feature tree represents the relationships between these different feature primitives, and is constructed using information from the lists. It can be then broken down into elements which correspond to actual machinable features. In the Autolisp program feature tree construction is accomplished by repetitive searches of the feature primitive lists, and the implementation of recursive programming.

The information output by the interface program is a machining document and a set of machining contours. The machining document is a complete set of instructions to the system operator for the machining of the prismatic part using Mastercam. Machining contours are modified sections of the part geometry saved in the form of IGES standard files.

The system cannot handle tolerancing information. Also, the fact that the system copes with features located on the top or bottom sides of a 2 1/2D component only, further restricts its flexibility. Its universal appeal is also limited due to the fact

that it uses as its input Autocad's internal drawing database structure, and not an industry standard file format. Furthermore, its output has to be manually processed by an operator before a part can be machined, and has been tailored specifically to the Mastercam package, again restricting its application potential.

2.7.3 Vosniakos's approach

Vosniakos [Vosniakos and Davies 1990] presented a 2 1/2D prismatic part post-processor for interfacing a wireframe CAD modeller with the ICAPP process planning system. The post-processor uses the IGES file format for input, deriving new information when necessary and formalising it as Prolog facts. The product model catered for in the work consists of a wireframe 3D structure representing the prismatic part, and an accompanying set of projections constituting an engineering drawing of the model.

The process adopted follows the steps described below:

- Information about the component is read from the IGES file.
- Necessary data for the CAPP program are extracted.
- Product information is enhanced.
- Information is converted to a format accepted by the CAPP program.

After reading the component information from the IGES file, and discarding the irrelevant fields, data enhancement has to be performed. This consists of:

- Correction of coordinate values.

- Calculation of additional parameters for geometric entities.
- Creation of a 2D engineering drawing from the 3D model.
- Assignment of curves created by surface intersection to the generating surfaces.
- Derivation of links between geometric entities.

A separate module saves the information derived into appropriate ASCII files. The user is notified of unprocessed entities in an output message file.

The post-processor starts by reading line by line from the IGES file of a prismatic part until the record delimiter is encountered. The first decision to be made is that of the type of an entity. A unique title is given to entities possessing a unique combination of "entity type number", "form number" and "use number". The entities are next divided into ones needing further manipulation (eg geometric entities) and others not needing any (eg dimensions). Changes in entity coordinates due to the entity being defined in a "local" coordinate system have to be catered for. Additional information is needed in several cases for geometric entities and it has to be calculated. For lines the parametric line equation is derived, and for arcs and circles their plane equation. Also, apart from describing each entity, the system needs data on how these entities are linked to one another. Assumptions are made to simplify finding links between lines, arcs and linear curves.

Analytic geometry routines are used in many places to derive data or perform entity comparisons. These include:

- Definition of a line by two points, a point and a vector or two intersecting planes.
- Definition of a plane by three points or a point and a vector.
- Definition of a point by a line intersecting a plane or an arc intersecting a plane.

Entities needing manipulation by more than one module of the post-processor are output to the appropriate files in the last step of the execution. Each entity becomes a Prolog "fact" consisting of a "functor" and "arguments". The functor is the entity's title and the arguments are its particulars. Coordinates are represented as three real numbers in a list.

Despite the method's effort at interfacing CAD with ICAPP for 2 1/2D prismatic parts, some "awkward" points of the IGES standard made impossible the transfer of the full context of a product model unless a series of restrictions were applied, thus limiting the system. These awkward points, and other system limitations include:

- too loose a definition of some entities. For example, even for a very simple entity like the line, multiple mapping is possible into the IGES line, or copious data, or even composite curve entities.
- entity coordinates are real numbers and are thus subject to rounding errors. When it comes to model transfers, rounding errors do happen. IGES pre-processors are sometimes the cause of such errors due to the different representation of the same entity in the two linked systems, circular arcs being well-known examples.
- apart from rounding errors, it is not uncommon to find logical errors as well.
- the linking patterns assumed by the system for geometric entities, were derived from rules describing what is considered a valid wireframe model. Variations however are common in wireframe modelling.
- the relationship of the entities after the data transfer was inferred by the system according to certain design rules, which are not universally applicable.

2.7.4 Zhao and Baines's approach

Zhao and Baines [Zhao and Baines 1993, Zhao et al. 1993] describe an interface designed to link the prototype CCSPLAN generative process planning system developed by the authors [Zhao and Baines 1993, 1992, 1992] with Autocad using the DXF file format. The system accepts as input the DXF files of 3D wireframe models of prismatic or rotational components. CCSPLAN needs the components to be defined as a set of related machinable surfaces, therefore a conversion process has to be performed on extracted DXF data to represent them as machinable surfaces in a coded format (as required by the coding system in CCSPLAN).

This process involves four stages-modules:

(1) *Data pre-processing*. 3D edge and vertex model data from the DXF input file are first redefined according to adjacent and incidence matrices. This is done so that coincident points can be easily identified based on the number of adjacent vertices or incident edges to each vertex. Coincident points are vertices which may have more than three incident edges, and they have to be removed from the wireframe model in order to extract surfaces. These points are thus identified and flagged for later use.

(2) *Graph definition*. A wireframe model can be represented by 2D graphs suitable for extracting surface information. The vertices and edges in a graph correspond to those in the wireframe model. To simplify programming, the wireframe model can be deliberately broken down into several sub-models, each being represented by a graph of not more than 8 nodes. This stage decomposes the model stored in the

adjacent and incident files into sub-models, each representing a solid block or hollow spaces. It then automatically verifies each sub-model based on certain developed relations, and converts the sub-model into a graph, stored in graph files.

(3) *Boundary extraction.* Surface, loop and intersection boundaries are next extracted from the graphs representing the sub-models. Because the part has been represented with simple graphs of not more than 8 nodes, extracting a boundary is in fact a straight-forward search procedure based on data in a simple tree structure.

(4) *Surface recognition.* Surface boundaries have to be distinguished from the other types of boundaries, and their orientation specified before they can be used by the CCSPLAN system. The task is accomplished by this module, but it is not completely automatic. The user is prompted to provide the type of one boundary in each sub-model (or graph). The system will then identify the types of the remaining boundaries based on the information given by the user. Surfaces represented by surface boundaries are next transformed into a plane, and described by their normals. The direction of the normals (achieved by the right-handed rule with the boundary and the normal) yields the required orientation of the surfaces.

With the above approach, surfaces are only extracted according to the input data format of a particular manufacturing system (CCSPLAN). The authors report that more investigation is required to combine the extracted surfaces into features for practical process planning.

2.7.5 Zhang's interpreter

Y. Zhang [Zhang 1991, Zhang and Mileham 1991] developed an interpreter, again using the DXF file format for input, able to extract features from symmetrical rotational part drawings. The interpretation process consists of the following stages:

(i) The input DXF file is read and only the necessary entity information extracted. This includes lines and arcs, along with their corresponding coordinates, as well as dimensioning and tolerancing information. For a symmetrical rotational part, one view of its drawing provides complete manufacturing information. Further data enhancement is performed to remove those geometric entities from the database which are below the centre line (since they are symmetrical with the ones above it). The remaining entities are then used for the identification process.

(ii) The entities forming the external profile of the component are identified. This is formed by joining pure geometric machining surfaces (eg faces, arcs etc). The reasoning process for entity identification is based on characteristics unique to the profile of a symmetrical rotational component. Once the external profile entities are identified, they are removed from the data file and the entities forming the internal profile (if any) are next processed.

(iii) Grooves (if any) are extracted from the external profile. In manufacturing, a groove and thread are treated as special features and machined using special form cutting tools. Therefore, to construct a manufacturing feature based product model, grooves and threads must be identified separately. During this stage, the system first

checks each recess. Based on its dimensional parameters, it is determined whether the recess is a groove or not. If a groove is extracted, the remaining profile is rearranged (sequence number, feature types, dimensions). This procedure continues until all recesses are checked. The next step is to extract the threads with their body numbers and location information from the database. The process is repeated for the internal profile, if any.

(iv) Tolerances are allocated to their relevant dimensions. The system deals with dimensional tolerances only. Two conditions are used to identify and allocate each tolerance entry to its relevant feature. The first, termed the *size condition*, specifies that the basic size of the tolerance should be equal to the relevant dimension. The second, termed the *position condition* is based on the information of the dimension's witness lines, particularly the tolerance string position and the witness gap. A feature's relevant dimension and tolerance can be extracted if it satisfies both the previous conditions.

(v) Identified features and their parameters are output in a format appropriate for process planning. Each machining feature occupies a record of the data file. The record contains the sequence number, feature type, dimensional parameters and tolerances. The data file can be easily accessed for further processing or modification.

2.7.6 Other approaches and general comments

Apart from the approaches described so far, an identifier able to extract geometric information from a part data file (generated using a 2D wire-frame CAD modeller) has also been developed [Wang and Lin 1987]. However, information such as tolerance, surface roughness etc is still typed interactively. Henderson [Henderson 1984] developed a methodology to extract part features from 3D CAD data. However, only geometric feature information is processed. Another approach, also able to extract geometric features from 3D CAD models and represent them in numerical as well as symbolic form is reviewed in [Bond et al. 1988]. The geometric extractor, AUTOGEM, has been developed to interface a specific CAD system named CADAM, and is not applicable generically. Mortensen and Belnap [Mortensen and Belnap 1989], describe a methodology employing feature recognition for rotational parts from a 2D CAD system database (although an interactive graphics system was used to generate the test data). This system was limited none the less, as it lacked automatic dimension extraction.

The STEP protocol has been formulated by the International Standards Organisation to be the future standard in data transfer [ISO STEP 1988, Danner 1990, Mason 1991, Shah and Mathew 1991]. However, it is not yet widely accepted by industry, and is incompatible with the majority of the current generation of commercial CAD systems [Sivayoganathan et al. 1993].

Another solution to the complex problem of identifying and extracting manufacturing features from CAD databases, is to build special CAD systems. These

utilize features for their internal component description databases, thereby bypassing the problem of interpretation altogether. Several attempts have been made at developing "feature-based" CAD systems [Gao and Case 1992, Dixon 1988, Shah and Rogers 1990] with considerable success. These were examined in more detail in section 2.6.3. However, these bespoke systems have found very limited use commercially (they are not applicable universally). Also, it is still not possible for these CAD systems to store all the detailed technical information required for a variety of different applications without losing their generality as design systems. Therefore, information mapping (conversion) between these design systems and the related application systems is inevitable [Gao and Case 1993, Shah et al. 1988].

Although all these approaches have contributed significantly to this domain, research on the topic is far from being conclusive. It is considered that more effort should focus on the following areas:

(a) Automatic interfacing is often limited by the fact that only a specific CAD modeller can be used. A CAD neutral format (widely accepted, industry standard) specification should therefore be employed to overcome this restriction. Any CAD modeller can then be used, as long as the modeller can convert drawing data to the neutral format. This will ensure a wide system flexibility and universal application potential. The system output format should also not be restricted to a particular application domain, but should be recorded in a rather more generic, neutral format for the same reasons.

(b) Geometric as well as technological information (such as tolerances) should be

included in the data interpretation process, in order to provide all the necessary data of a component to a CAPP system.

(c) Features located on any side of a prismatic component (and not just the top or bottom), as well as feature interactions should be investigated.

CHAPTER 3

INFORMATION PHILOSOPHY OF THE SYSTEM

3.1 Introduction and General Overview of BUCADIP and BUFIP

This work presents a proposed system, BUCADIP (Bath University CAD Interpreter for Prismatic components) and BUFIP (Bath University Feature Identifier for Prismatic components), developed by the author at Bath University. The system is intended to automate the data transfer from any conventional wireframe-based CAD system (using the DXF industry standard file format) to a CAPP system for prismatic components. It has been designed to generate feature-based component description output files, to be used directly by a CAPP system, or easily edited manually if required.

The data transfer process consists of four main stages:

(i) The CAD DXF file of a typical three view engineering drawing of a prismatic component is used as the system input, constructed to a specific convention. In this way compatibility is retained with existing engineering practices. In addition, since three view engineering drawing wireframe models can be readily produced by virtually all CAD surface and solid modellers, the techniques developed in this work can be applied to feature recognition from such models as well.

(ii) The DXF file is interpreted by BUCADIP, which outputs a condensed and

refined file that can be used for feature recognition.

(iii) Features are then extracted from the interpreted file by algorithms embedded in BUFIP. The approach adopted resembles that of *rule-based template matching* used for solid model feature identification, and described in detail in section 2.6.2. Templates are defined for various flat and cylindrical manufacturing features. These are applied by BUFIP to each view of the drawing, using information from the interpreted file. Various developed algorithms subsequently match the templates against actual drawing entities. If a match is found, the appropriate feature, along with its associated characteristics and dimensions is extracted. The system is capable of successfully identifying several combinations of interacting features.

(iv) The results are recorded in a data file using the ASCII standard format for further processing or retrieval by a CAPP package. Use of the ASCII standard for output, as well as descriptive statements for the features and their attributes, facilitates manual checking or editing of the output file. More importantly, it also retains compatibility with a wide variety of CAPP systems, since it is not restricted to the specific input format of a particular system.

The system, which is entirely automatic in its operation, has been built with expandability and ease of maintenance in mind, hence a modular structure has been used, including modules (using developed assorted algorithms) for:

- Sorting and classifying a component's data entities.
- Identifying multiple interacting flat features (through slots, steps and pockets).
- Identifying multiple cylindrical features (holes, stepped holes and threads).

- Assigning tolerances to the component's overall dimensions.
- Determining the minimum required raw material block shape for the component to be machined.
- Identifying and reporting a comprehensive set of possible user errors encountered during interpretation.

Compared with other established methods for feature recognition (described in detail in **Chapter 2**), this approach addresses and resolves their main shortcomings, namely:

- The absence of any tolerance allocation methods.
- The absence of possible interactions between various features.
- The possible locations of features on one or two sides of a component only.
- The absence of a common industry standard input/output format.

The methodologies developed from this research can also be generalised and applied to similar but distinct problems in other application domains. For example, the developed general feature recognition algorithms could assist in drawing retrieval from CAD databases. Drawing files incorporating specific features could thus be easily located. Another example is part classification and coding. This is concerned with identifying the similarities among parts and relating these similarities into a coding system, an operation still mostly performed manually today. The developed strategies could (in conjunction with a coding system) classify and assign codes to a workpiece according to the types and characteristics of features it contains, its block size and tolerance finish etc.

A further divergent application example of this research work is NC program generation. BUFIP produces manufacturing feature based output files, with all the parameters included for locating and machining the features. This means that a program designed to produce NC codes could take advantage of this information, and generate all the NC codes required for machining the part automatically, thus greatly improving productivity. The research work's algorithms could also be applied to evaluation of product designs. By using BUFIP to quickly and automatically extract and display the manufacturing features of a particular product and their characteristics, its design can be evaluated and optimised in terms of machining facility requirements, production costs etc.

The BUCADIP and BUFIP system, although developed for strictly research and proof-of-concept purposes, was envisaged to eventually be capable of operating in a small to medium sized company batch machining environment. It could thus bring enhanced possibilities for automation to the large number of small to medium-sized enterprises currently using simple CAD draughting systems, and whose majority of design data exist in the form of engineering drawings. This intention affected the choice of hardware and software development platforms, as outlined in the following section. Details of the research system are provided in the following sections and chapters.

3.2 Development Software and Hardware Selection

Once the objectives and aims for the research were determined, the choice of

the development programming language became critical. Choice was eventually narrowed down to either FORTRAN or C. After consideration, C was selected, offering more power, flexibility, speed of execution and versatility than FORTRAN.

C is often referred to as a middle level language, lying somewhere between assembler (low-level) and PASCAL (high-level) [Schildt 1990]. Part of the reason that C was invented was to give the programmer a high-level language that could be used as a substitute for assembly language. As it is known, assembly language uses the symbolic representation of the actual instructions executed by the computer. There is a one-to-one relationship between each assembly language instruction and the machine instruction. While this relationship makes it possible to write highly efficient programs, doing so is quite tedious and error-prone. On the other hand, high level languages such as PASCAL, are greatly removed from the machine. A statement in PASCAL has virtually no relationship to the sequence of machine instructions that is ultimately executed. However, while C retains high-level control structures, such as are found in PASCAL, it still allows the programmer to manipulate bits, bytes and addresses in a way more closely tied to the machine rather than the abstraction presented by other high-level languages. For this reason, C has occasionally been called "high-level assembly code". Because of C's dual nature, it allows programmers to create very fast, efficient programs without having to resort to assembly language.

The philosophy behind C is that the programmer knows what he or she is doing. For this reason, the C language almost never "gets in the way of" the programmer, and one is free to use (or abuse) the language any way one sees fit. The

reason for this "programmer as king" approach is that it allows a C compiler to create very fast and efficient code because it places the responsibility for error-checking on the user [Borland International Inc. 1991]. However, the drawback with this is that it makes C a difficult language to master, with users requiring a long amount of time and experience to learn how to use it effectively.

Following selection of the programming language, the issue of the appropriate hardware platform for development was raised. Two main platforms were considered as capable of handling the tasks involved:

- a UNIX operating system based workstation
- an MS-DOS operating system based IBM-PC compatible.

Following careful consideration of the merits and disadvantages of each system, it was decided to use the IBM-PC compatible solution, since a modern computer of this category has more than adequate performance for the intended research work. PC compatibles are also widely employed by the vast majority of small to medium manufacturing enterprises, the envisaged eventual users of the system. An Intel 486-processor based system was thus selected, with Borland's Turbo C++ v.3.0 as the development programming language.

Following a review of available CAD packages for IBM-PC compatibles, it was decided to adopt AUTOCAD for the purpose of designing sample prismatic components to be processed by the CAD interpreter module under development.

AUTOCAD is a wire-frame CAD modeller with significant 3D capabilities

and the ability to remove "hidden" lines from drawings [Autodesk Inc. 1986]. It is a powerful tool for both drafting and design. Some of the reasons for AUTOCAD's power are its wide range of commands, its large library, third-party tie-ins and the AUTOLISP programming language that it supports, enabling the user to customise the package. These features, along with many others, have established AUTOCAD as the industry standard CAD package for PC compatibles.

3.3 General Structure of the System

The developed system consists of two main programs: BUCADIP and BUFIP.

BUCADIP (Bath University CAD Interpreter for Prismatic components) is the first program and is invoked for the interpretation of engineering drawings containing prismatic component data. It is an autonomous, self-contained program with no modular structure, since it is relatively small. It accepts as input the DXF data file of a prismatic component drawn using standard engineering drawing practices, according to British Standard BS308 (first angle projection). The logic could however be modified to handle other draughting conventions. A few enhancements for clarification purposes need to be incorporated into the drawing, these are discussed in more detail in section 3.4 below.

BUCADIP's main function is to act as a pre-processor for BUFIP. A DXF file of a component can be quite large, even for simple parts (the DXF file format in general is described in more detail in **Chapter 4**). This is because it contains a lot

of redundant data, not necessarily useful for feature identification or process planning. BUCADIP effectively "filters" the DXF file, stripping and recording only the data necessary for complete feature identification, tolerance allocation and subsequent process planning of the component. The resultant interpreted file (in ASCII format) is much smaller, clearer, and can be easily edited manually if required for the additions or substractions of entities.

It could be argued that this filtering and interpreting of a DXF file could have been implemented in the main body of the system (BUFIP), thus eliminating the need for having to run two programs. However, keeping the initial filtering process separate from the feature identification process, and using an (intermediate in effect) interpreted file presents a significant advantage: the system could easily be modified to handle other standard CAD file formats (eg IGES) by changing only the BUCADIP program to accept different data input and still produce the same interpreted file format for feature identification. It also allows for easier manual error-checking should the need arise. BUCADIP is currently in its ninth version (v.9.0), and is discussed in detail in **Chapter 4**.

BUFIP (Bath University Feature Identifier for Prismatic components) is the feature identification program. This forms the main body of the research work, and is structured in a modular way, not only because of its large size, but also due to ease of maintenance and upgradeability reasons. Several classes of manufacturing features are grouped in separate modules, hence one can expand the possible identifiable feature list by adding enhancements to existing modules, or simply creating new ones. Each module forms a separate C program executed by the main

control module under specific circumstances (for example the holes module is executed to identify any possible holes and their parameters).

In brief, BUFIP currently consists of the following module-programs:

1. **Main.C** Main part of the program. Also contains the main interpreted file reading functions of the project.
2. **Common.C** Contains functions common to all entities (lines, circles, arcs) like their layer name and description.
3. **Props.C** Contains functions specific to storing entity properties.
4. **Slots.C** Contains the multiple through slot identification algorithms.
5. **Steps.C** Contains the multiple step identification algorithms.
6. **Holes.C** Includes holes and threads identification algorithms.
7. **Pockets.C** Contains the pockets identification algorithms.
8. **Tolrance.C** Identifies and assigns tolerances to the component's overall dimensions. Also determines the required raw material block shape envelope.
9. **Msg.C** and **Msgtable.C** Provides the comprehensive error reporting handled by the system.
10. **Prjdefs.H** and **Msgdefs.H** Header files assigning various project and message definition parameters.

A graphic depiction of the various BUFIP modules can be seen in **Figure 3.1**. The system accepts as input BUCADIP-interpreted DXF files (having an .INT extension) and consecutively applies various feature identification algorithms in order

to extract and record a wide range of manufacturing features and their combinations. The results are logged in a feature-based component description file in ASCII format. The file is structured in such a way as to be capable of being used directly and automatically by a CAPP system, as well as being easily checked and edited manually if required. The whole interpretation and identification process can be seen graphically in **Figure 3.2**. Detailed descriptions of the various BUFIP modules (currently in their ninth version v.9.0), as well as typical application examples using the system follow in **Chapters 5, 6, 7** and **8**.

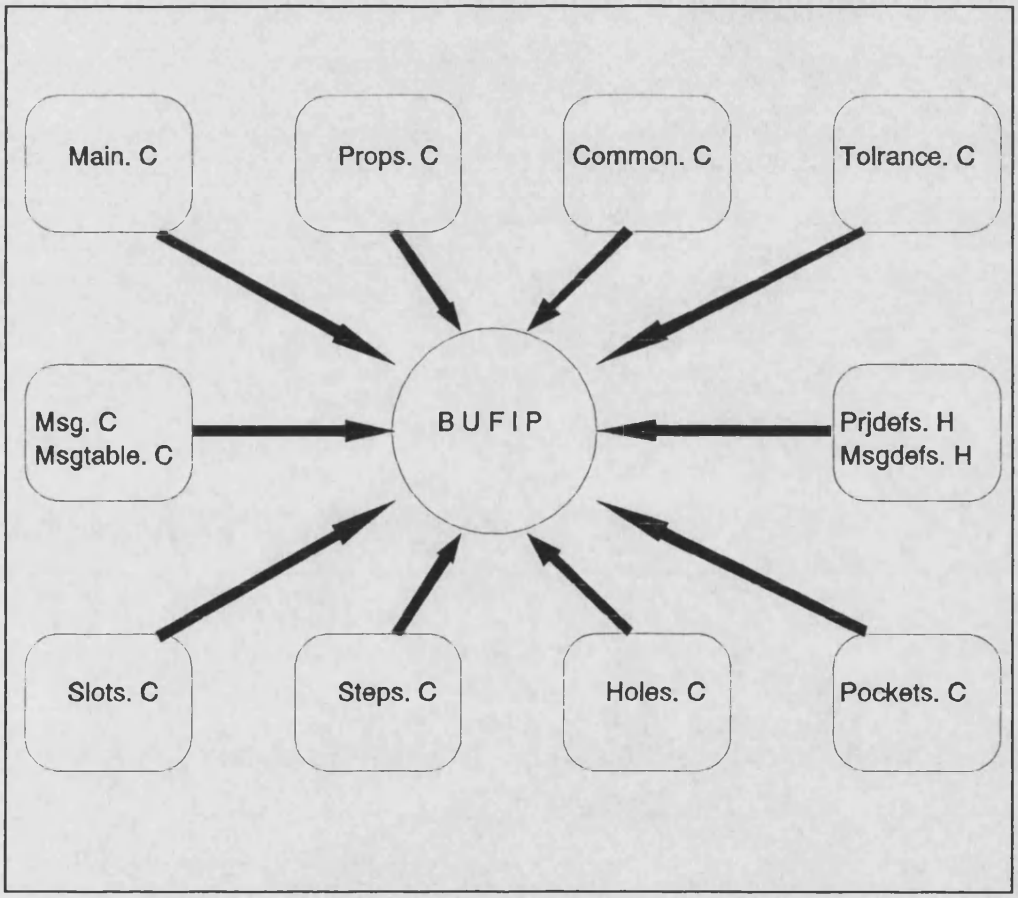


Figure 3.1 The constituent modules of BUFIP

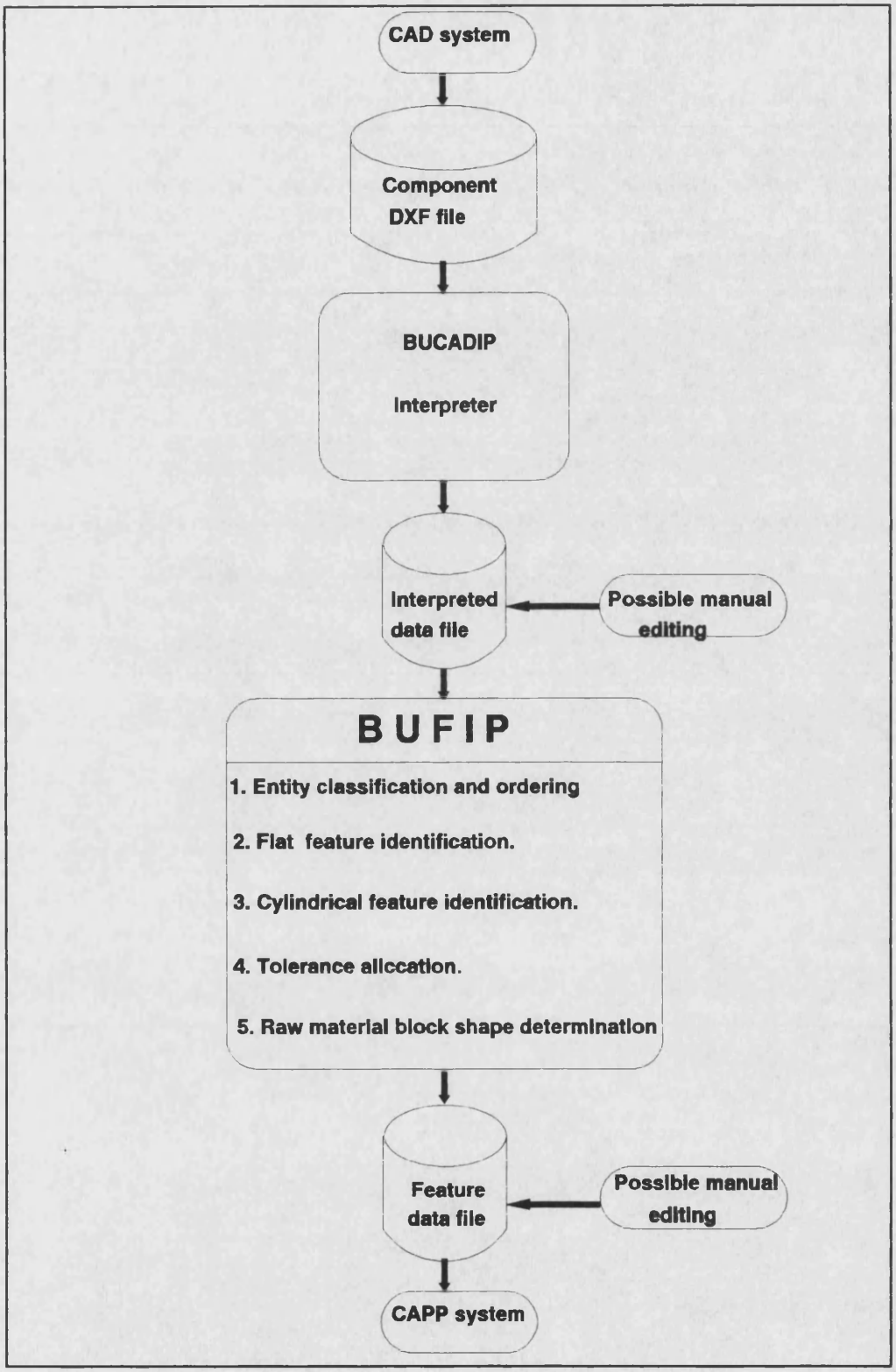


Figure 3.2 General interpretation process structure

3.4 The Layered View Approach

For a prismatic part, complete manufacturing information can be provided by a three-view drawing in 2D space using orthographic projection (standard engineering drawing). Three views are not always either necessary or sufficient for the unambiguous representation of a 3D object (more views could be necessary for clarification). However, such exceptions are considered to be outside the scope of this research. It is assumed here that three views are provided and that they determine a unique component.

Given three views of the same component however could unnecessarily complicate the interpretation process, since there is a duplication of the same features on different views of the drawing. This can be avoided by placing the different views of the drawing in different *layers* in the CAD database when drawing the component. A layer is similar to a self-contained mini CAD data file: only the entities assigned to it are contained in there. Since the layer that an entity belongs to is also reflected in its entry in the DXF file, the feature identification task becomes simpler. This is because drawing entities are already allocated to their respective views. The use of prescribed layers is the main requirement the system expects from the user and it is not considered unusually demanding (the practice of layering CAD drawings is widely used in industry). The views are layered in a predefined convention as follows (there is no current standard for layering views of a drawing):

Layer 0	FRONT view
Layer 1	PLAN view
Layer 2	END view

The layer names or numbers can be (and indeed are) used interchangeably by the system. A typical drawing of a component indicating the layered view approach can be seen in **Figure 3.3**.

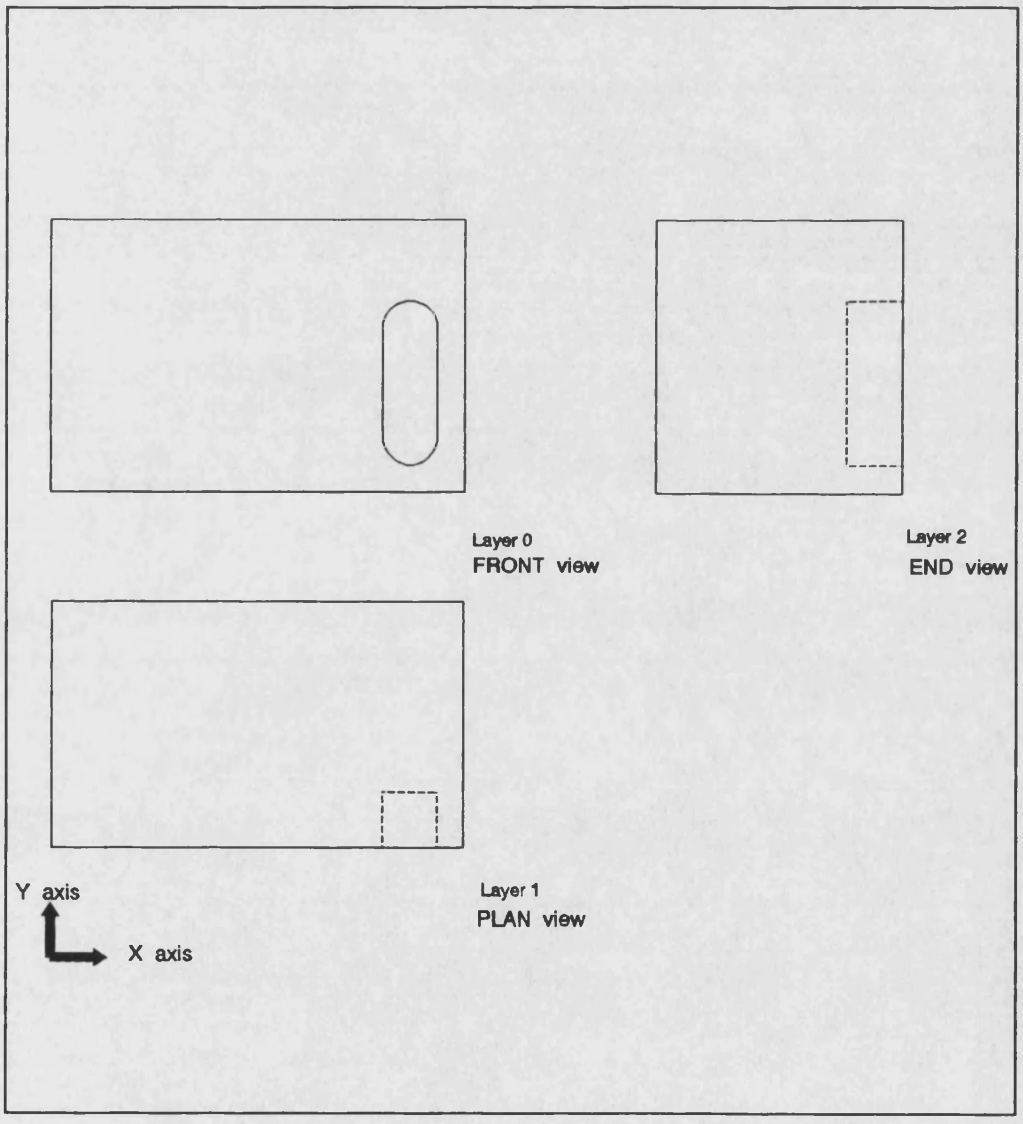


Figure 3.3 The layered view approach to a drawing

Furthermore, the requirement for extracting and assigning tolerances to the component's overall dimensions necessitated the introduction of three more layers:

Layer 3	DIMF (DIMensional Front view layer: containing dimensioning and tolerancing information for the front view).
Layer 4	DIMP (DIMensional Plan view).
Layer 5	DIME (DIMensional End view).

It could be argued that a single DIMensional layer would suffice for assigning tolerances to a component's overall dimensions, and this would be valid. However, as mentioned previously, the system was designed with considerable future expansion potential, and it was envisaged that individual feature tolerance allocation would eventually be incorporated (unfortunately time limitations prevented this implementation). Hence the definition of three dimensional layers, one for each view. The system however is perfectly capable of operating with the three primary views only (0, 1 and 2) without any dimensioning - in this case it will not assign any tolerances.

Because the layering system proposed above could initially confuse potential users, **Appendix A** provides some step-by-step guidelines for configuring AUTOCAD before drawing for use in such a way.

CHAPTER 4

BUCADIP : THE CAD INTERPRETER

4.1 Introduction

BUCADIP (Bath University CAD Interpreter for Prismatic components) is the preprocessor for the new system. It uses as input the DXF data file of the engineering drawing of a prismatic component. The data file is filtered and reprocessed to preserve only the information necessary for feature identification, tolerance allocation and subsequent process planning of the component. This is recorded in an easily read ASCII format structure for subsequent processing.

The interpretation and filtering process can be quite complex, due to the intricacies of the DXF files data format.

4.2 The DXF File Format

In a CAD system, the drawing database is stored in a system specific, very compact format in order to maximise performance, and minimise disk space and memory usage. However, the need to allow data exchange between different CAD packages led to the requirement for a commonly accepted format for drawing information interchange.

Autodesk [Autodesk Inc. 1986], the developers of AUTOCAD, defined such a format and named it the "Drawing Interchange File" (DXF) format. Data files using this format are termed DXF files and are encoded in ASCII characters and numbers, so that they can be easily read directly by other programs. Most wireframe-based CAD packages on the market soon adopted this format (since it was developed by the industry leader) and incorporated the ability to both read and produce DXF files. Thus the DXF format quickly emerged as the widely accepted industry standard for CAD data exchange.

However, due to its ASCII nature, a DXF file tends to be very large (even simple drawings can have DXF files of tens of thousands of bytes) and contains a lot of redundant information for process planning purposes. This can be seen more clearly if the DXF file structure is considered.

A DXF file is composed of four main *sections* containing a multiplicity of *groups*, each of which occupies two lines in the DXF file. The first line of a group is a *group code*, which is a nonnegative integer output in FORTRAN "I3" format (that is, right justified and blank filled in a three character field). The second line of the group is the *group value*, in a format which depends on the type of the group as specified by the group code. The specific assignment of group codes depends upon the item being described in the file.

Variables, table entries, and entities are described by a group code that introduces the item, giving its type and/or name, followed by multiple group codes that supply the values associated with the item. In addition, special group codes are

used for file separators such as markers for the beginning and end of sections, tables, and the file itself. Entities, table entries, and file separators are always introduced with a 0 group code that is followed by a name describing the item. An example DXF file (much abbreviated) with explanatory comments is shown in **Figure 4.1**.

The four main sections of the DXF file are:

(1) The HEADER section containing general information about the drawing. Each parameter has a variable name and an associated value. These variables are set with various commands from within AUTOCAD, and include items such as the AUTOCAD version number, units formats, helping drawing grid layout, viewpoints etc.

(2) The TABLES section containing tabled definitions of named items such as fonts, styles and line types. Each table is introduced with a 0 group code with the label "TABLE". This is followed by a 2 group code naming the table (eg "STYLE") and a 70 group code that specifies the maximum number of table entries that may follow. The tables in a drawing may contain deleted items, but these are not written in the DXF file.

Following the header for each table are the table entries. Each entry consists of a 0 group code identifying the entry type, a 2 group code giving the name of the table entry, a 70 group code specifying flags relevant to the table entry and additional group codes that give the value of the table entry. The end of each table is indicated by a 0 group code with the value "ENDTAB".

```
0                   (Begin HEADER section)
SECTION
2
HEADER
.....<continue>.....
ENDSEC              (End HEADER section)
0                   (Begin TABLES section)
SECTION
2
TABLES
0
TABLE
.....<continue>.....
ENDTAB
0
ENDSEC              (End TABLES section)
0
SECTION             (Begin BLOCKS section)
2
BLOCKS
.....<continue>.....
0
ENDSEC              (End BLOCKS section)
0
SECTION             (Begin ENTITIES section)
2
ENTITIES
0
LINE                (Line entity)
8
PLAN                (Layer name)
6
DASHED              (Line type)
10
      29.150000          (Xstart coordinate)
20
      7.2500000          (Ystart coordinate)
11
      39.150000          (Xend coordinate)
21
      23.100000          (Yend coordinate)
.....<continue>.....
0
ENDSEC              (End ENTITIES section)
0
EOF                 (End of file)
```

Figure 4.1 An example DXF file (much abbreviated)

(3) The BLOCKS section containing Block Definition entries (blocks are groups of drawing entities, for example lines, circles, arcs), describing the entities comprising each block in the drawing. The format of the entities in this section is identical to those in the ENTITIES section described below. All entities in the BLOCKS section appear between BLOCK and ENDBLK entities. These appear only in the BLOCKS section and are never nested (that is no BLOCK or ENDBLK entity ever appears within another BLOCK-ENDBLK pair). Blocks of entities are not currently supported by the interpreter program (it is assumed that entities are drawn one-by-one), although they could be incorporated in the future without major modifications.

(4) The ENTITIES section containing information of all the drawing entities, such as geometric entities (lines, circles etc) and text (dimensions, tolerances etc) including any block references. The ENTITIES section is the most important part of the DXF file for the interpreter program, since it contains the data defining a component. Each entity type is uniquely specified by an entity type name following a 0 group code. Its characteristics such as coordinates, layer name etc are defined by specific group codes and values following its type name.

The following gives the format of each entity currently supported by the interpreter. Some group codes that define an entity always appear, and some are optional and appear only if they differ from their default values. In this case, the group codes the interpreter needs always appear. It should be noted that the group codes describing various entities do not necessarily occur in the order given below, but in the order they were drawn (which can be completely random). The interpreter takes this into account, making no assumptions about the order of entities, and

ignoring any group codes not currently supported.

The names and parameters used for the supported entities are given in the list that follows. Every entity contains an 8 group code that gives the name of the layer on which the entity resides, and a 6 group code showing the line type (CONTINUOUS for continuous lines, DASHED or HIDDEN for "hidden" line-features on the opposite side of the component from the viewing point). Both group codes are vital for effective feature recognition by the system. The rest of the group codes that make up the various entities are:

LINE	10	X start point
	20	Y start point
	11	X end point
	21	Y end point
CIRCLE	10	X centre
	20	Y centre
	40	Radius
ARC	10	X centre
	20	Y centre
	40	Radius of curvature
	50	Start angle (measured in an anti-clockwise direction from the X-axis horizontal plane)
	51	End angle (ditto)

SOLID (Used to describe arrows in tolerance extension lines)

Three points defining the corners of the arrow, with the last being the tip:

10	X1	
20	Y1	
11	X2	
21	Y2	
12	X3	(arrow tip X)
22	Y3	(arrow tip Y)

TEXT (Used for tolerancing)

10	X insertion point
20	Y insertion point
1	Text (tolerance) value

A graphical depiction of the main sections of the DXF file format can be seen in **Figure 4.2.**

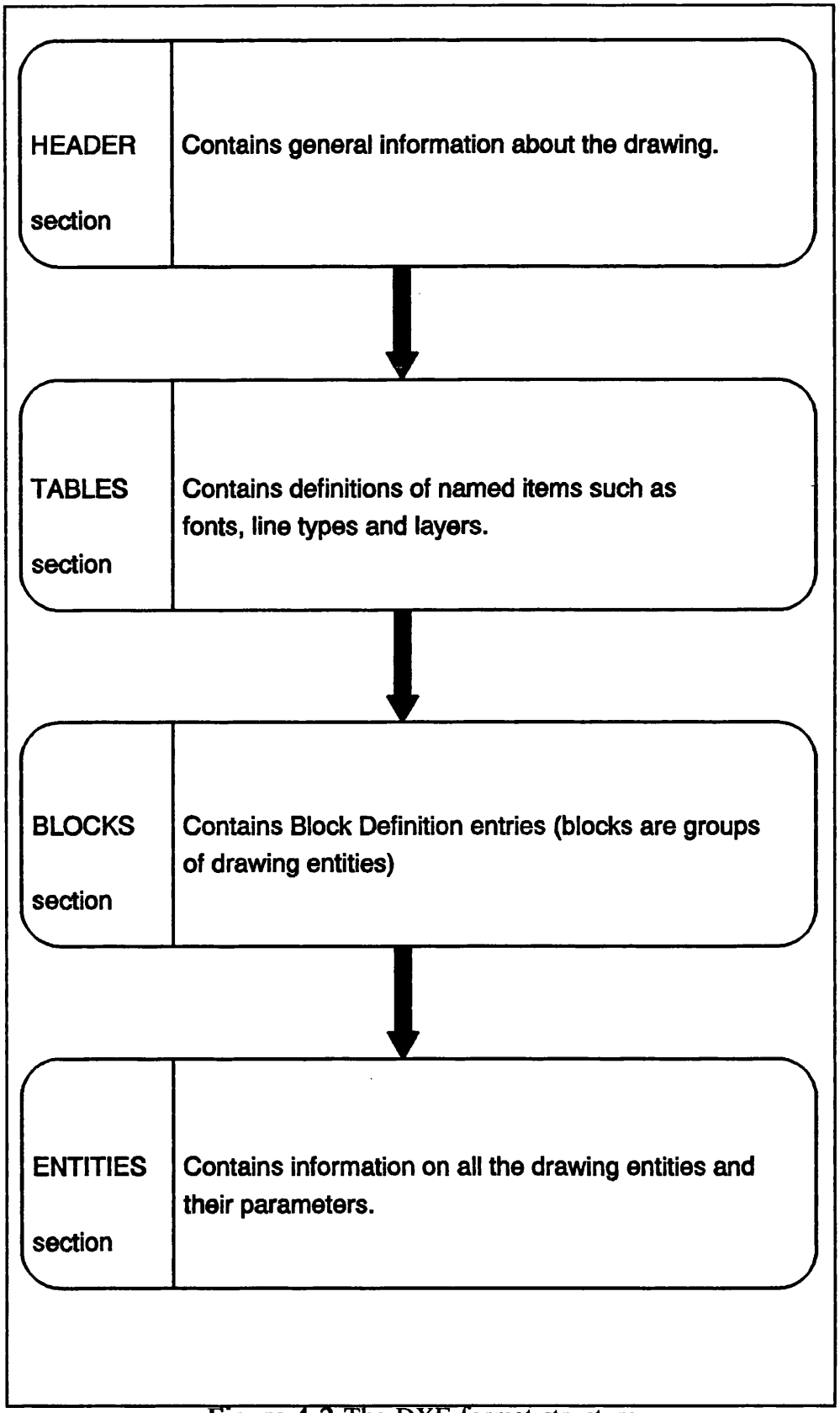


Figure 4.2 The DXF format structure

4.3 Part Drawing Preparation, Precautions and Common Errors

In order to ensure good, consistent interpretation of prismatic component drawings (as well as effective subsequent feature recognition) care was taken to adhere to some general guidelines on part drawing creation. This helped to avoid errors being reported from the main feature identification program (the interpreter program is very fault-tolerant: it will simply interpret the information it comes across). As is usually the case, the old computing principle of "garbage in garbage out" applies here as well.

As mentioned previously in **Chapter 3**, the drawing should be prepared using standard engineering drawing practices according to British Standard BS308 (first angle projection). Hidden entities (and features) should be drawn using dashed instead of solid notation. AUTOCAD uses the "HIDDEN" and "DASHED" line types for this purpose. However, when drawing on a standard resolution on a small screen, the DASHED and HIDDEN line types can become virtually indistinguishable from their CONTINUOUS counterparts. This led to the common pitfall, observed during experimentation, of drawing continuous entities which were meant to be dashed, and vice versa, especially when modifying a drawing. Users should take care to ensure that the right kind of line type is selected by frequently checking the properties of the entities being drawn.

Another potential mistake could occur when drawing the edges of a component. The system expects the component's edge profile to be continuous (that is the end point of an entity forming the starting point for the next), otherwise it

reports an error. Sometimes, particularly when editing existing drawings, it is not uncommon to end an entity approximately (but not exactly) where the next entity starts, thus creating an error. Users can avoid this by using a helping grid and the "snap" command for attaching entities exactly.

The layered view approach, as described in **Chapter 3**, should be implemented for each drawing. **Appendix A** contains step by step guidelines for configuring AUTOCAD for this purpose. Once properly configured, users should ensure that they are drawing the correct entities on the correct layer. Although sounding obvious, it takes some familiarisation for an inexperienced person to adapt to this technique. It was observed during experimentation that sometimes entities belonging eg to the plan layer were being drawn by accident on the front layer (but in their correct position on the plan view) thus leading to several errors reported during feature recognition. Since AUTOCAD always indicates the current layer being drawn on a corner of the screen, this pitfall can easily be avoided.

Care should also be taken for the same reasons when dimensioning and tolerancing. For effective tolerance allocation, AUTOCAD should be properly configured, and the dimensioning layers (mentioned in **Chapter 3**) installed. The guidelines for properly configuring AUTOCAD for tolerancing are described in **Appendix A** and involve toggling certain AUTOCAD variables to certain values.

The above precautions and potential problems during drawing preparation are highlighted in the special component drawing example shown in **Figure 4.3** below.

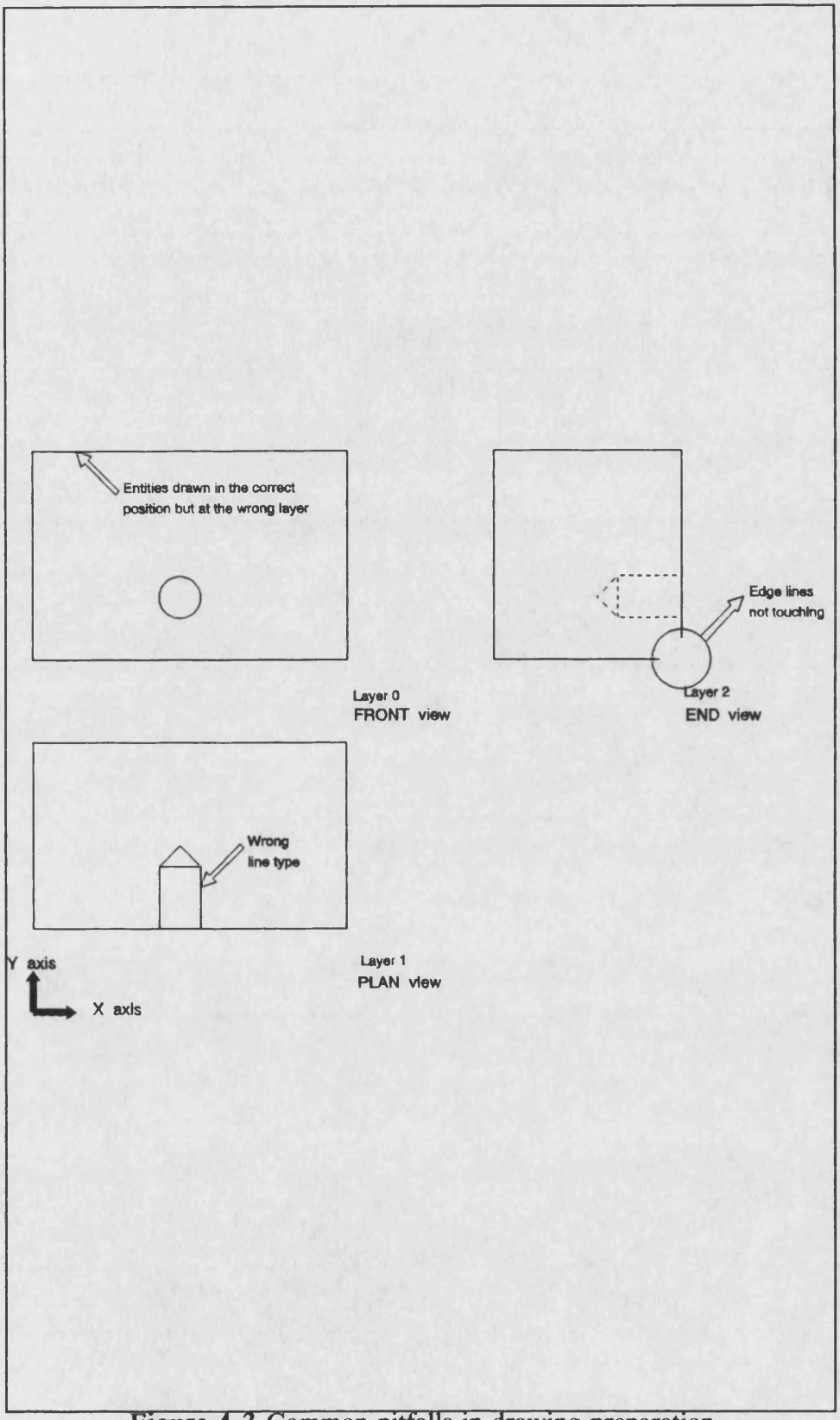


Figure 4.3 Common pitfalls in drawing preparation

4.4 The Strategy for Interpretation

Once a drawing is properly prepared and saved in the DXF format, the filtering and interpretation process can begin. This is achieved by running the BUCADIP program from the MS-DOS command prompt in the form:

C:\>BUCADIP9 filename.DXF > filename.INT

where filename is the actual name of the drawing file. This has as a result the creation of the interpreted file with the extension .INT (INTerpreted). The general procedures that the program follows to produce an interpreted file can be depicted in a flow chart form, as shown in **Figure 4.4**.

The program starts by defining the various entity types and assigning them an integer number for subsequent processing. Hence, the following entity type definitions are made:

LINE	0
CIRCLE	1
ARC	2
SOLID	3
TEXT	4

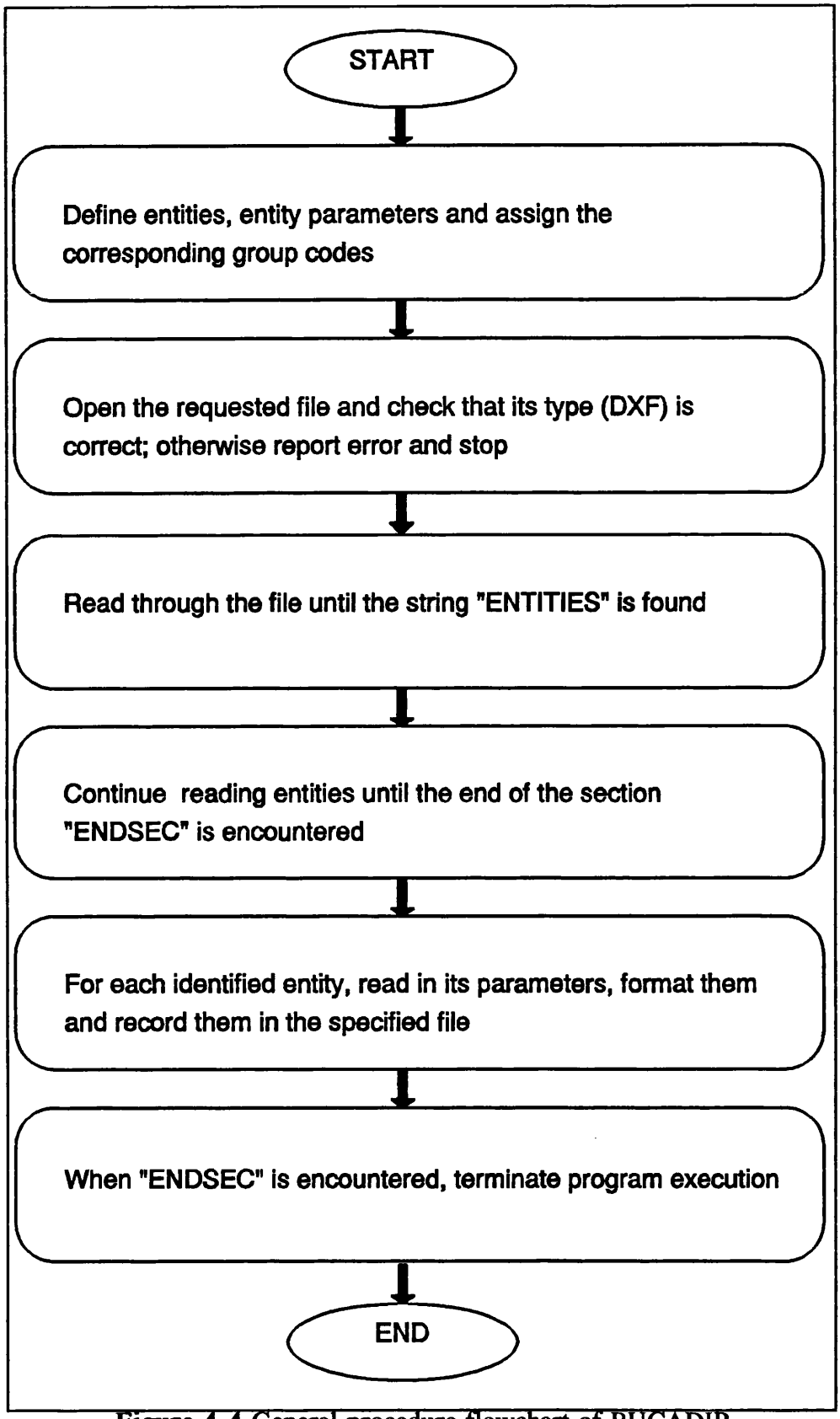


Figure 4.4 General procedure flowchart of BUCADIP

Next, the entity parameters are defined and the appropriate group codes assigned. Some parameters are general (like layer name), some are entity specific (like arrowx) and some apply to multiple entities. The following entity parameter definitions are declared:

Entity par.	Group code	Comments
LAYER	8	Layer name
LINETYPE	6	Continuous, dashed or hidden
XSTART	10	Line start and end point coordinates
YSTART	20	
XEND	11	
YEND	21	
XCENTRE	10	Circles & arcs centre coordinates
YCENTRE	20	
RADIUS	40	Circle radius or arc radius of curvature
STARTANGLE	50	Angle from which arc starts
ENDANGLE	51	Angle to which arc ends
ARROWX	12	Tip of arrow - X coordinate
ARROWY	22	Tip of arrow - Y coordinate
STRING	1	Tolerance text entity value

The program then attempts to open the requested file. It first checks that the file is a DXF one by checking its extension. If the extension is not .DXF, the program aborts with the message:

"Usage: bucadip9 filename.dxf"

The contents of the file are next checked. If the file is empty, if the file name is typed incorrectly or if the file contains inappropriate data, the program stops with the following error message:

"Error: cannot open (filename). Please ensure the file name and type is correct."

thus indicating to the user what is wrong, and asking for another attempt.

Once the correct file name is typed and the file opened, the program starts

reading through the file, looking for the string "ENTITIES" that indicates the beginning of the entities section. This is achieved by implementing a defined function *getgroup*. The *getgroup* function reads the DXF file line by line, strips the carriage return (newline) characters, and returns the group code (*grpcode*) and corresponding value (*grpvalue*). This can be a string (like LINE, CIRCLE etc) or a numerical value (like an X coordinate, say 10.342000).

Once the string ENTITIES is identified, indicating the beginning of the entities section, the program continues reading in lines (using the function *getgroup*) until the end of the section is encountered, with the string "ENDSEC". This time though, the program also implements another defined function, *describe*.

The function *describe* forms the main core of the interpreter program. Its task is to identify the various supported entities and their associated parameters, group them, classify them and record them in the specified output data file using specific formatting. To accomplish this it makes use of the C command *switch*, which allows the program to select a specific action from a range of choices depending on certain conditions.

Function *describe* starts by initialising (setting to 0) all entity parameter variables. Entity parameter variables are defined with the small case equivalent of the entity parameters themselves (eg variable yend for entity parameter YEND etc). It then proceeds to assign to individual entity parameter variables, their appropriate values using the *switch* command and the corresponding group code. Hence, for example, in the case of XSTART (group code 10 as mentioned earlier) the read value

(say 25.345000) is assigned to variable xstart, and so forth. It should be noted that certain values are not numeric but character-based (eg line types or layer names) in which case the strings themselves are copied to the appropriate character based variables (eg "CONTINUOUS" copied to linetype and so on).

Finally, the second main part of function *describe* is activated. This actually records the identified entities and their associated parameters in a specific format to the indicated output file. The exact format for each entity is discussed in detail in section 4.5 below. The *switch* command is used again to select the appropriate output format depending on the identified entity (for example in the case of ARC, the ARC formatting is recorded on file). When the end of the ENTITIES section is reached (indicated by the string "ENDSEC"), the program finishes execution and the interpreted file has been produced.

4.5 Output Format of the Interpreter

For each identified type of entity, the interpreter uses a specific format for recording it in the output file. These styles of formatting were streamlined with each successive version of the program, for easier understanding and reading of the interpreted file by both human users or computer programs.

As mentioned previously, the resultant file is produced in standard ASCII format, hence enabling easy manual editing should the need arise. Descriptive language statements are used for the various entities and their associated parameters

(rather than group codes or numbers), thus greatly facilitating editing or cross-referencing the interpreted file with the original drawing for verification purposes. The output format used for the various entities and their associated parameters is as follows:

For line entities:

"(linetype) LINE on layer (layer) from
START (xstart, ystart) to
FINISH (xend, yend) "

where (linetype)=the line type (CONTINUOUS, DASHED or HIDDEN)
(layer)=the layer name or number (eg PLAN)
(xstart, ystart)=start point coordinates
(xend, yend)=end point coordinates.

For circle entities:

"(linetype) CIRCLE on layer (layer)
CENTRE (xcentre, ycentre)
RADIUS (radius) "

where (xcentre, ycentre)=centre coordinates
(radius)=the circle radius.

For arc entities:

"(linetype) ARC on layer (layer)
CENTRE (xcentre, ycentre) RADIUS (radius)
START ANGLE (startangle) END ANGLE (endangle) "

where (xcentre, ycentre)=the arc's centre coordinates

(radius)=the radius of curvature of the arc

(startangle)=the relative angle where the arc starts (from AUTOCAD's zero)

(endangle)=the relative angle where the arc ends (from AUTOCAD's zero).

For dimensioning line arrows:

"(linetype) ARROW on layer (layer)

POINT (arrowx, arrowy) "

where (arrowx, arrowy)=the coordinates of the arrow's tip.

For tolerancing text:

"(linetype) TEXT on layer (layer)

(text) "

where (text)=the actual tolerance text value.

An actual sample of the interpreted file of a typical prismatic component can be seen in **Figure 4.5** below:

```
CONTINUOUS LINE on layer FRONT from
 START (120.000000,190.000000) to
 FINISH (120.000000,140.000000).
CONTINUOUS CIRCLE on layer PLAN
 CENTRE (70.000000,40.000000)
 RADIUS (10.000000)
CONTINUOUS ARROW on layer DIMP
 POINT (20.000000,120.000000)
CONTINUOUS TEXT on layer DIMP
 -1.0000
DASHED ARC on layer PLAN
 CENTRE (100.000000,80.000000) RADIUS (10.000000)
 START ANGLE (0.000000) END ANGLE (180.000000)
```

Figure 4.5 Sample of an interpreted file

CHAPTER 5

SETTING UP THE SYSTEM FOR FEATURE IDENTIFICATION

5.1 Introduction

Once a prismatic component drawing is properly drawn and interpreted, it is the turn of BUFIP (Bath University Feature Identifier for Prismatic components) to extract and classify a wide range of manufacturing features.

As it was mentioned previously in **Chapter 3**, BUFIP (which forms the main body of the research work) consists of several separate programs structured in a modular way. These include elements for setting the environment for correct and effective feature identification, as well as for detecting and extracting a comprehensive set of features and their combinations.

The first three sections of this chapter provide a more detailed description of the way the system's operating environment has been configured and how it executes the various modules for effective and structured feature identification. The remaining chapters complete the general overview by describing the detection, sorting and classification algorithms that have been implemented for the various manufacturing features.

It should be stressed that due to the complexity (even in its modular form) and

size of the program (more than 4,000 lines of code), a thoroughly detailed, statement-by-statement description of its operation is impossible within the confines of this thesis. Hence, a more generalised approach is followed, where the implemented algorithms are discussed in more detail rather than their actual code description, although this is also covered in more general terms. If specific reference to the actual program code is required, a complete program listing, available as a separate report, can be consulted [Linardakis 1995].

5.2 Operation Sequence of the Core BUFIP Module (main.c)

All the various programs of the BUFIP system are integrated in the central module **main.c**. This forms the core element, where the other modules and their associated functions are attached and executed according to a specific operational order.

Program **main.c** starts by incorporating and executing the **prjdefs.h** and **msgdefs.h** header files. **Prjdefs.h** contains a large number of variable and data structure definitions used by several other modules. **Msgdefs.h** contains the error message code definitions. The programs **msg.c** and **msgtable.c** are also implemented at this stage. Together they activate the system's comprehensive error reporting message facilities, described in more detail in section 5.4.

Next, **main.c** opens the requested interpreted file and checks its validity. Once this has been verified, it starts reading the file line-by-line. For each entity

encountered, its line type is first determined (CONTINUOUS, DASHED or HIDDEN). The entity type itself is next identified, along with the layer that it belongs to, using **common.c**.

For each identified entity, **main.c** reads in its associated parameters, classifies them and stores them in the appropriate data structure. For this purpose it passes control to **props.c**, which is responsible for correct data classification, as well as for implementing various data sorting algorithms (discussed in the following chapters). In the case of line entities, **main.c** reads in their coordinates, while **props.c** undertakes further manipulation.

The custom coordinate system algorithm (discussed in detail in section 5.5) is next implemented using **common.c**. This transforms all entities' coordinates relative to component edges for the three views of a drawing.

Following the custom coordinate system implementation, the component's edge profile is next determined using algorithms in **props.c** (discussed in detail in **Chapter 6**). These not only identify the line entities that shape a component's edge profile for each view, but also sort these entities into a consistent order, and determine where each side of the component starts and ends.

Finally, the feature identification modules of the system are executed by **main.c**. These perform specific feature identification operations on the sorted data structures using several developed algorithms, which are discussed in detail in **Chapters 6,7**. More specifically, the order of execution of the various feature

identification modules by **main.c** is:

slots.c

steps.c

holes.c

pockets.c

tolrance.c

Once these programs are run, and the various identified features and their characteristics recorded in an output file, the system ends operation. A summary of the general operation sequence of **main.c** can be seen in a flowchart form in **Figure 5.1.**

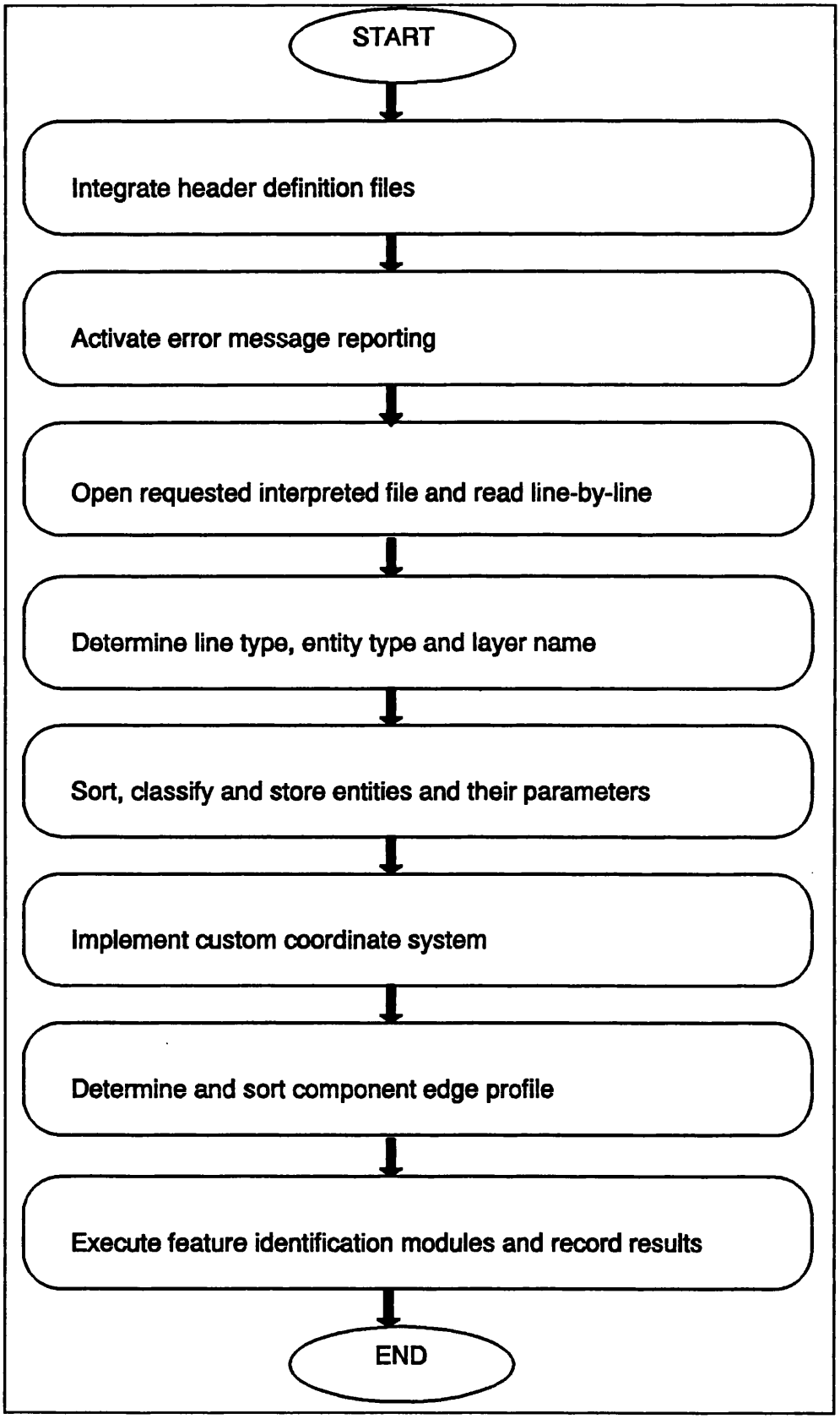


Figure 5.1 General operation sequence of **main.c**

5.3 Classification of the System Operating Environment by Functions and Programs

In the previous section, a general overview of the operation sequence of the core of BUFIP, **main.c**, was presented. This section shows in more detail (with brief explanatory comments), the various modules of the system, along with their associated functions, used to sort and prepare data for feature identification:

1) MAIN.C

The core element of BUFIP.

Supported functions :

minmax (establishes maximum and minimum points for each layer).

getcoord (grabs X, Y coordinates of line entities).

2) PRJDEFS.H

Defines maximum number of objects per layer.

Defines layers and layer values.

Defines data structures for Arrows, Text, Circles, Lines, Layers, Edge profiles, and Arcs.

Defines some functions' names and type.

3) MSGDEFS.H

Defines abbreviation codes for all messages, both for feature and error reporting.

4) COMMON.C

Contains functions common to all modules.

Supported functions :

ExtractLayer (extracts the layer name and description for each entity).

ResolveCoordinates (implements the custom coordinate system).

5) PROPS.C

Contains functions for storing the various entities and their parameters.

Supported functions:

StoreEdge	(determines the edge profile of the component for each layer and also identifies the start points of the different sides for each layer's edge profile).
SetEdgeLine	(sorts and stores separately the lines forming the edge profile and also determines if the corners of a component's profile are real or "virtual").
StoreLineProperties	(sorts and stores line entity parameters and stores the minimum and maximum points of each layer).
StoreCircleProperties	(reads in, classifies and stores circle parameters).
StoreArcProperties	(reads in, classifies and stores arc properties and also determines the arcs' start and end point coordinates).
StoreTextProperties	(reads in and stores text entities).
StoreArrowProperties	(reads in and stores arrows' points coordinates).

6) MSG.C

Returns message code number for selecting the appropriate message from msgtable.c

Supported function:

Msg (used by several modules for selecting appropriate messages).

7) MSGTABLE.C

The actual messages in a listed form.

Figure 5.2 shows the operation sequence of **main.c** in a pseudocode flowchart form based on the functions and modules described above.

The BUFIP program runs automatically from the MS-DOS command prompt. The command for activating it must have the following form:

C:\>BUFIP9 filename.INT > filename.FET

where *filename* is the actual name of the drawing file. This has as a result the creation of the processed output file with the extension .FET (FEaTure based).

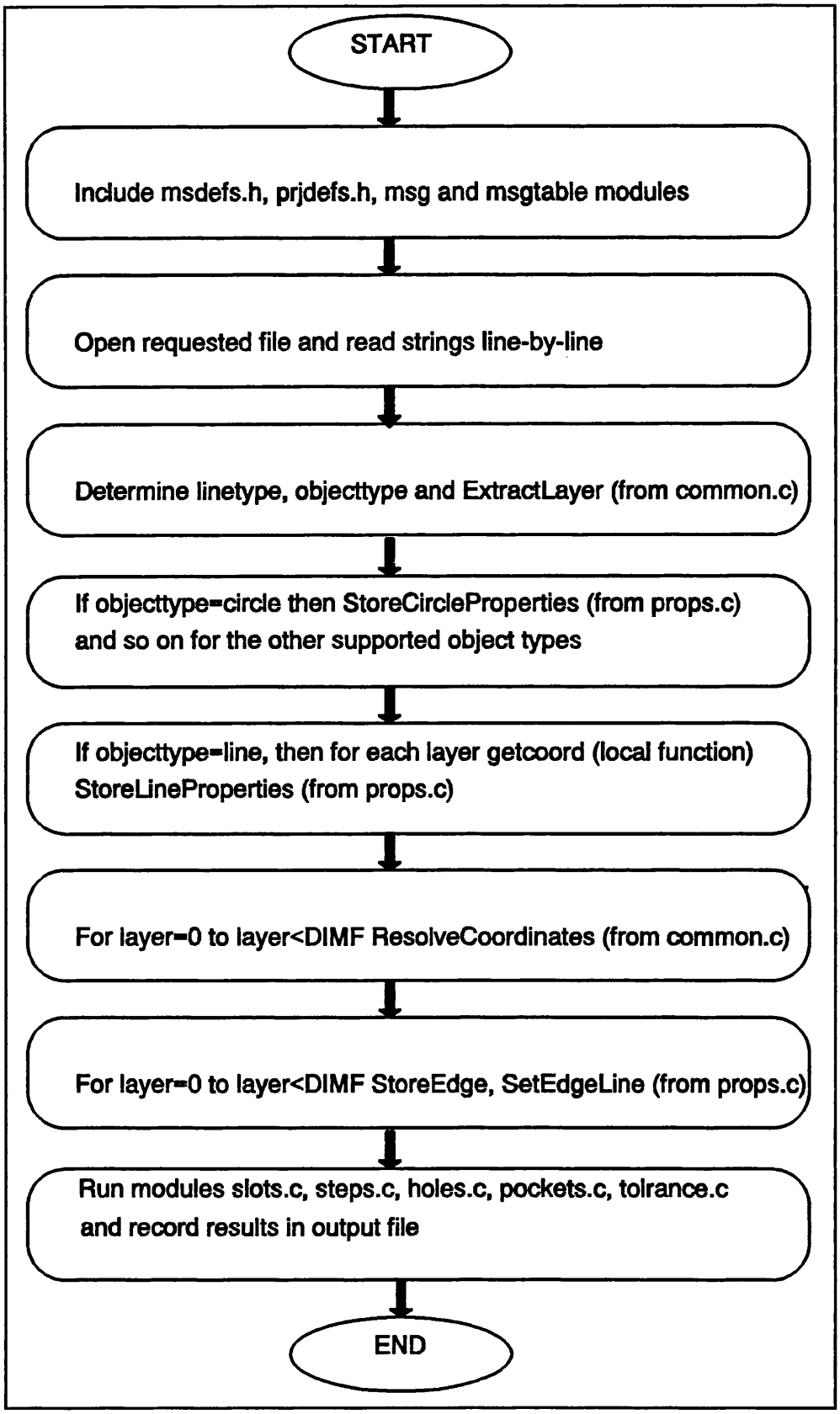


Figure 5.2 Program and function based operation sequence of **main.c**

5.4 Error Detection and Reporting

As mentioned previously, the system employs a comprehensive error detection and reporting system. This enables it to cater for a wide range of malfunctions, and alert the user to their cause, while at the same time attempting to identify features and complete its task without halting. This of course depends on the nature of the error encountered.

The header file **msgdefs.h** contains the various error code definitions, while the program **msg.c** implements the *msg* function used throughout the modules for selecting and flagging the appropriate error message. The error messages themselves are located in a separate program, **msgtable.c**. This has been done in order to facilitate the addition of extra error messages in the future. The system currently flags the following error messages:

- "Usage: bufip9 (filename).int".

Shown if the user incorrectly types the name, or filename extension of the program.

- "Cannot open file (filename)".

Shown if there is an error in opening the requested interpreted file (for example if the file format is wrong).

- "Unexpected EOF on file (filename)".

An unexpected End Of File has been encountered in the requested file.

- "Format error on file (filename) at line (linenumber)".

Some entity formatting error has been encountered at the specified line

number of the requested file.

- "Cannot resolve object description direction".

Certain object parameters are not described at the correct order.

- "Unresolved feature on layer (layername)".

A feature has been encountered at the specified layer that cannot be properly identified.

- "Cannot resolve perimeter on layer (layername)".

The system has difficulty in identifying the component edge profile for that particular layer.

- "Unresolved line type at line (linenumber)".

The line type at the specified line number cannot be identified.

- "Unresolved object type at line (linenumber)".

The entity type at the specified line number cannot be identified.

- "Too many objects on layer (layername)".

More than the maximum defined number of objects that the system can handle have been encountered at the specified layer.

- "Unresolved hole (X centre=() Y centre=()) on layer (layername)".

A hole on the specified layer at the specified centre coordinates cannot be properly identified.

- "Unresolvable angle on arc on layer (layername)".

The start or end angle of an arc on the specified layer cannot be decided.

- "Unknown pocket type".

A pocket has been encountered which cannot be identified.

- "Unresolvable tolerance on layer (layername)".

The tolerance value of a dimension on the specified layer cannot be decided.

5.5 Implementing the Custom Coordinate System

By default, AUTOCAD's implemented coordinate system has its point of origin on the bottom left hand corner of the drawing paper size used (**Figure 5.3**). While this makes it convenient for drawing entities on screen, it is unacceptable for process planning purposes, because a corner of the component is usually selected as the datum point of origin.

To cater for this perceived inadequacy, it was decided to implement a custom coordinate system in BUFIP. This assumes as its point of origin the bottom left hand corner of the component for each view of the drawing (**Figure 5.3**). Hence three points of origin are defined, one for each view. The various entities and their parameters for each view are then referenced relative to the point of origin of that view. This not only facilitates manual checking of the identified features and their characteristics, but also resembles more closely the actual selection and referencing of datum points for the component in an actual machining environment. Therefore it was felt that implementation of this custom coordinate system in BUFIP would prove beneficial for subsequent effective process planning.

However, the definition of this custom coordinate system does not preclude its conversion to another coordinate system with a different point (or points) of origin, if required. This would be straightforward to implement, by simply modifying the existing developed coordinate transformation algorithms.

The custom coordinate system is realized using the defined function

ResolveCoordinates in **common.c**. This is based on the following algorithm for transforming entity coordinates:

For each layer if Xmin,Ymin are the smallest identified X,Y coordinates, then:
Entity new X coordinate = Entity current X coordinate - Xmin
Entity new Y coordinate = Entity current Y coordinate - Ymin
and once all entities are transformed,
Xmin,Ymin = 0,0

This in effect references all entities on a particular layer relative to its minimum X and Y, which are then set to zero and hence become a point of origin.

In practice, function *ResolveCoordinates* is much more complicated since it has to cater for all the various types of supported entities and their associated parameters, as well as for the component's edge profile and its maximum layer points. Function *ResolveCoordinates* transforms the coordinates of the parameters of each entity type by successively incrementing a counter for each entity type (eg for i=0 to i<CircleCount; i++) and applying the transformation algorithm to all its parameters. In the end it transforms the maximum layer point and sets the minimum layer point coordinates to 0.

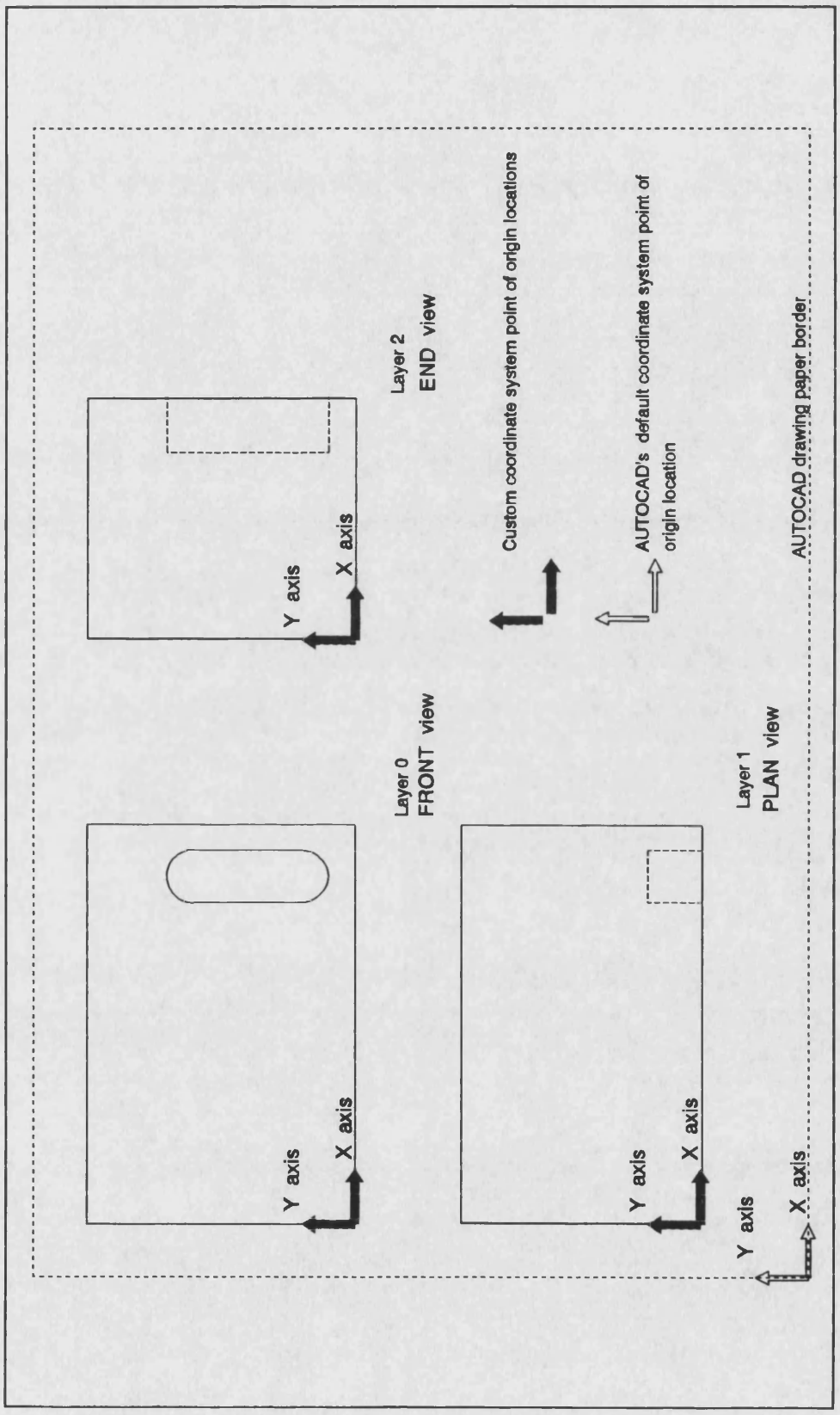


Figure 5.3 Implementation of the custom coordinate system

CHAPTER 6

IDENTIFYING FLAT MANUFACTURING FEATURES

6.1 Introduction

Following the preliminary setting up of the system for effective feature identification, the features themselves must then be extracted. For the purposes of compatibility with a locally developed CAPP system for prismatic components, it was decided to adopt Rustom's [Rustom 1992] classification of geometric features for prismatic components. This could make possible a future interfacing of the two systems for completely automated process planning of prismatic components. Besides, Rustom's feature classification is considered to be appropriate.

Rustom divides the geometric features of prismatic components in two distinct groups: flat features and cylindrical features. Both groups are further subdivided into basic and secondary features. Flat features consist of the basic flat face (no features) and the secondary step face, through slot and three types of pockets: open, side and closed. Cylindrical features consist of the basic plain hole, along with the secondary stepped hole, countersink and thread. **Figures 6.1** and **6.2**, indicate graphically and in tabular form the various features recognised by the system. This chapter describes in detail how BUFIP handles the identification of flat manufacturing features. The various feature identification algorithms are discussed, along with how the system configures and prepares the interpreted entities for the recognition task.

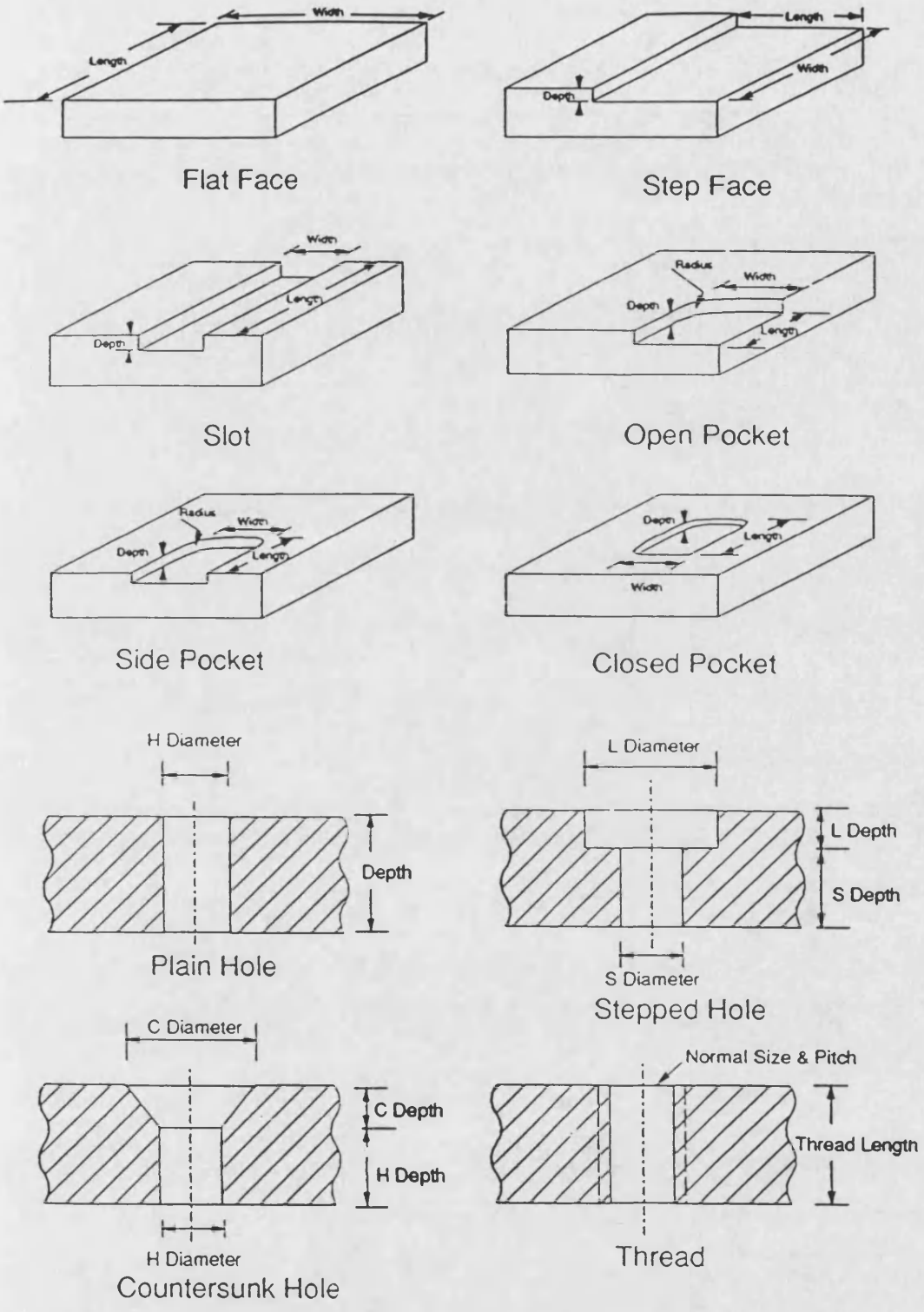


Figure 6.1 Types of features identified by BUFIP (after [Rustom 1992])

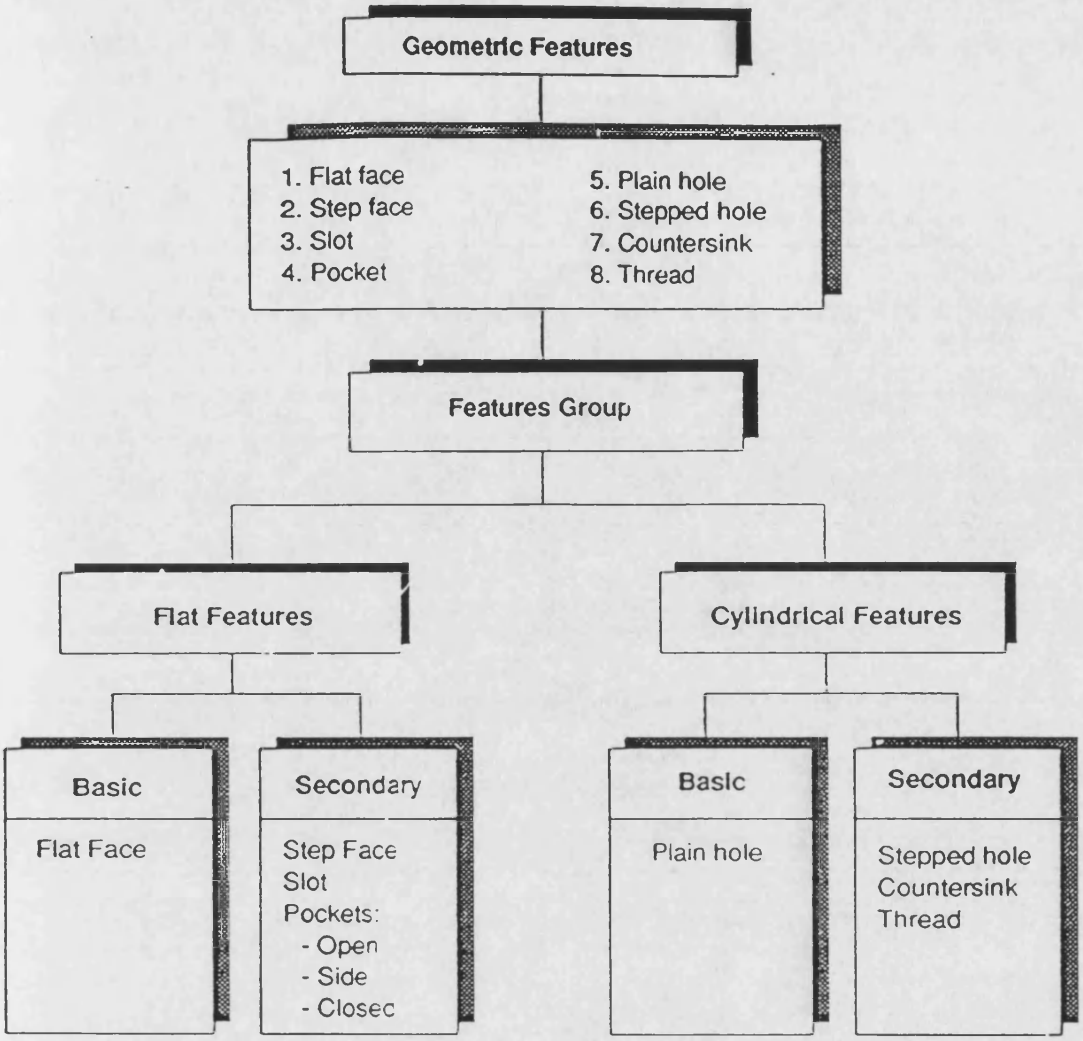


Figure 6.2 Classification of features according to Rustom [1992]

6.2 Determining the component's edge profile

Before attempting to resolve any flat features on a drawing layer, the system needs to determine which lines form the edge profile of the component for that layer.

In order to achieve this, function *StoreLineProperties* is first executed for each

read line. This classifies and stores the various line entity parameters. Function *StoreLineProperties* starts by ignoring "dot" lines. These are lines that start and end at the same point. It was found during experimentation that a user could inadvertently create such a line by double clicking with the mouse on the same point on the screen, usually when trying to position the finishing point of a proper line. This "dot" line could then go unnoticed (since it was overlapped by the proper line) and produce errors during feature recognition. Function *StoreLineProperties* compensates for this by ignoring any lines that have the same Xstart and Xend, as well as Ystart and Yend.

Next, function *StoreLineProperties* proceeds to ensure that lines are described in a consistent format. Hence, if the Xstart of a line is greater than the Xend, then its X coordinates are swapped around. Similarly, if the Ystart is smaller than the Yend, its Y coordinates are swapped around. This process results in all lines being described from left-to-right and top-to-bottom, as **Figure 6.3** shows, with the line directions being shown as arrows:

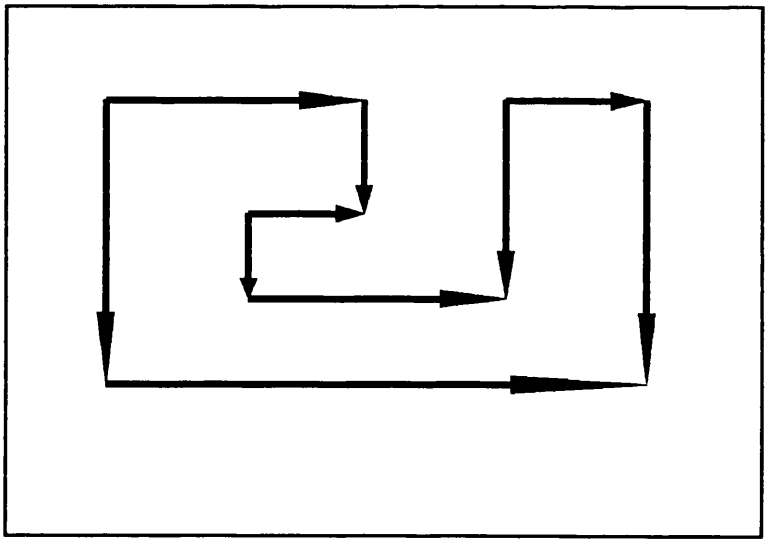


Figure 6.3 Left-to-right and top-to-bottom description

The minimum and maximum coordinates for each layer are next determined and stored. This is done by comparing the coordinates of each new read line against the currently established minima/maxima for the layer. If the coordinates are smaller/larger than the minima/maxima coordinates, then they become the new minima/maxima, and so on. Finally, function *StoreLineProperties* stores the determined minima/maxima for the layer, as well as each sorted line's parameters to an appropriate data structure.

Once the lines are sorted in the way described above, the component edge profile can then be determined. Before going into more detail of how this is achieved, it would be useful at this point to consider what constitutes this profile, and how it can be split into different sides for more accurate feature location reporting.

A component's edge profile for each layer is formed by continuous lines indicating the outline of the component for that layer (or view of the drawing). In its simplest form (for the purposes of this research), an edge profile will consist of just four lines, forming the four sides of the component for that view, as shown in **Figure 6.4**. Of course there can be components with three sides, indicating an inclined face, but these are excluded from the scope of this research. In fact, the system will report an error if a layer of any drawing contains less than four sides.

For the purposes of accurate flat feature location reporting, it was decided from an early stage to name and classify the four sides and four corners forming the general edge profile of a view of a component. The sides were thus named 0,1,2 and 3 in a clockwise direction from the top left hand corner (**Figure 6.4**). The corners

were similarly named corner 0,1,2 and 3 (**Figure 6.4**). Hence, if for example a through slot is found somewhere along the edge of a layer, it can be pinpointed accurately by stating that it belongs to, say, side 1 of the front view of the drawing, starting at coordinates X,Y (from the custom coordinate system). Similarly, if a step is found at a certain corner, it could be reported as belonging to eg corner 2 of the plan layer of the drawing. Therefore by using this technique, the various flat feature locations can be accurately determined.

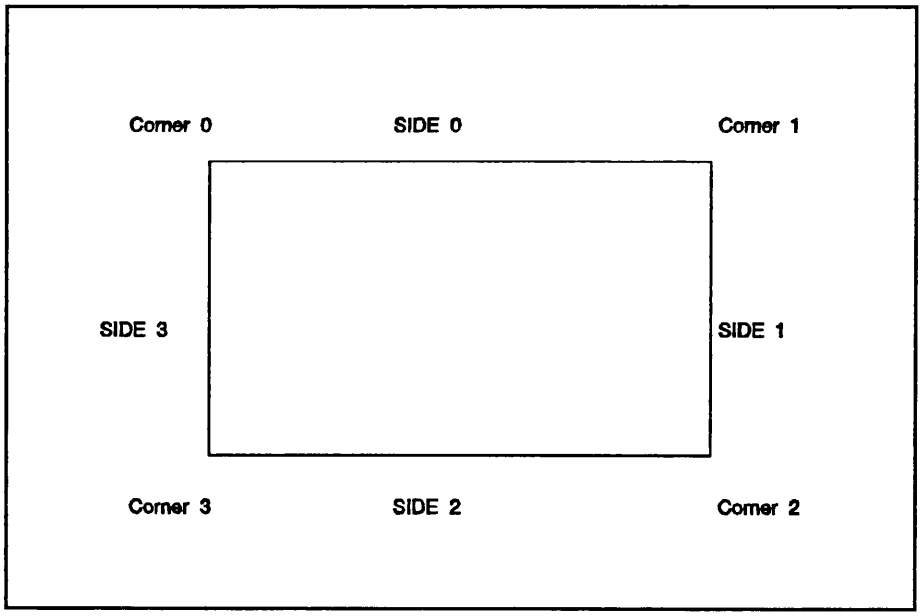


Figure 6.4 An elementary component profile

However, during development of the system, it was realised that while various simple features could be easily allocated as belonging to a certain side, complex profile geometries were a different proposition. Consider for example the edge profile shown in **Figure 6.5** below. In this ambiguous situation, does slot A belong to side 3 (left) or to side 0 (top) of the profile ? Clearly an algorithm would have to be developed that was able to decide under various situations how the four

component sides are formed, or in other words, where each side starts and where it ends.

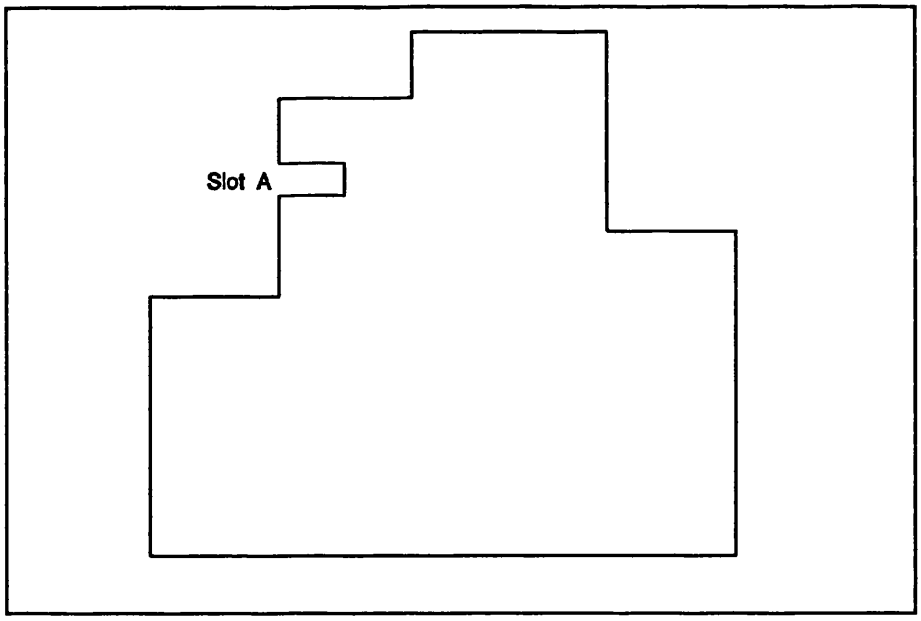


Figure 6.5 Ambiguous edge profile

After considerable experimentation with a wide range of component profiles, such an algorithm was eventually developed, and fine-tuned for handling a diverse variety of component profiles. The final version of the algorithm is as follows:

1. If there is not a real point at Xmin, Ymax, then the start of edge 0 is Xmin and the Y closest to Ymax.

2. If there is not a real point at Xmax, Ymax, then the start of edge 1 is Xmax, and the Y closest to Ymax.

3. If there is not a real point at Xmax, Ymin, then the start of edge 2 is Xmax, and the Y closest to Ymin.

4. If there is not a real point at Xmin, Ymin, then the start of edge 3 is Xmin, and the Y closest to Ymin.

Points can be either real or virtual. A "real" point in this context is termed a point where the component edge profile passes from, ie if there is a line arriving at or leaving from that point (otherwise the point has been termed "virtual". This concept is exploited later on for determining stepped faces).

Hence, applying this algorithm to components with four "real" corners, results in the correct positioning of the beginning and end points of the various sides on the appropriate corners, as shown in **Figure 6.6** (where the encircled points indicate where each side starts and ends).

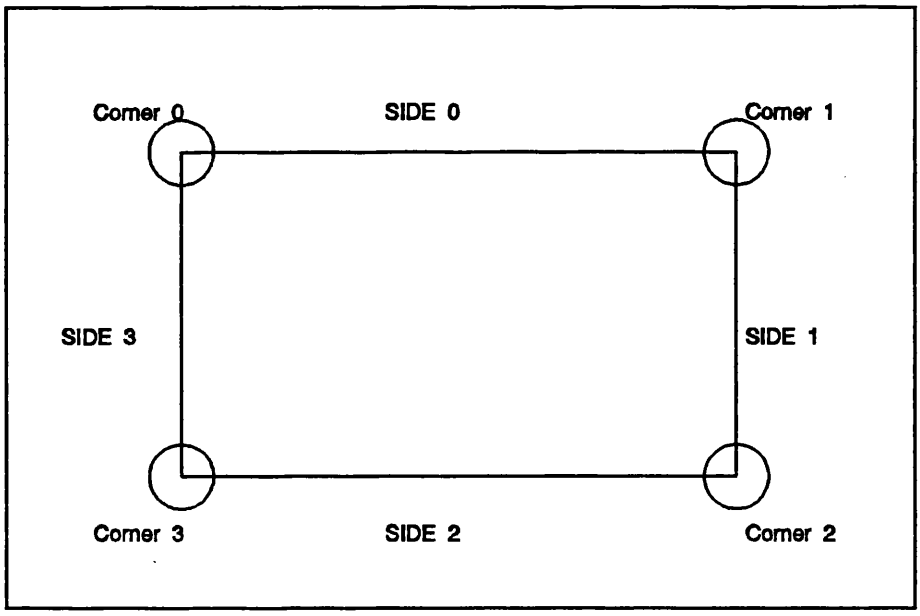


Figure 6.6 Elementary component profile with sides identified

At the same time, applying the algorithm to components with one or more "virtual" corners results in the positioning of sides in an acceptable, realistic and convenient manner. In practice, this means that complicated component profiles end up with 1-3 big sides full of features, and 1-3 smaller sides with hardly any features.

This accurately reflects actual process planning practices, where components are planned in a similar way, with the largest of the side(s) without features normally being used for locating the workpiece on the machine tool table.

Hence, by reconsidering the profile of **Figure 6.5** and applying the algorithm on it, slot A can be determined to belong to side 0, as **Figure 6.7** indicates. **Figure 6.7** also shows how side 0 includes all features to be machined (big side), while the other sides have no features at all, with side 2 (the largest of the plane sides) being used for the location surface.

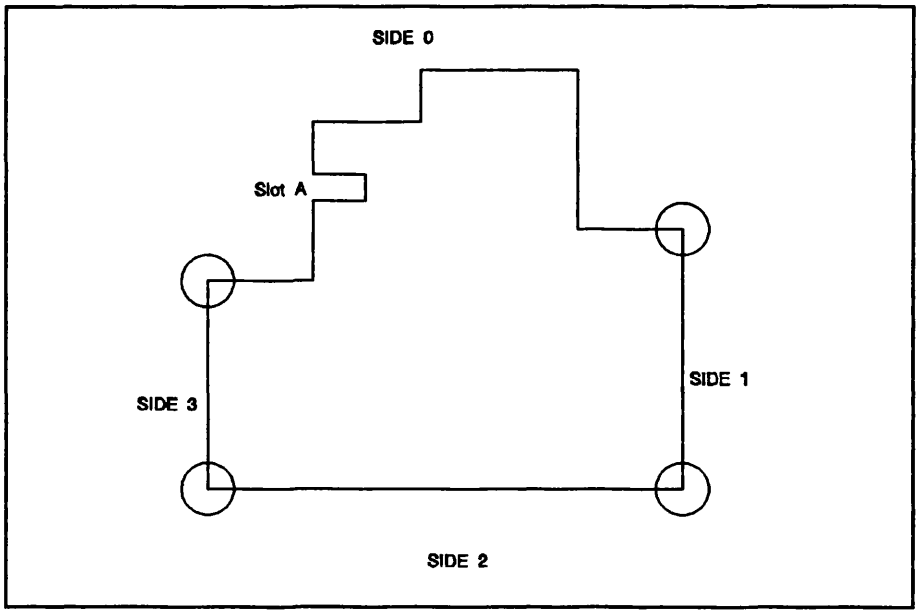


Figure 6.7 Ambiguous situation resolved

With this algorithm in place, various profile feature locations can now be accurately determined, as belonging to a certain side or corner of a particular view of the drawing. Summarising how BUFIP identifies and sorts a component edge profile.

Functions *StoreEdge* and *SetEdgeLine* from **props.c** are responsible for this task. Function *StoreEdge* performs the main processing required, with function *SetEdgeLine* providing auxiliary support. The functions are executed for each layer-view of a drawing.

Function *StoreEdge* starts by determining the horizontal line whose start is nearest to Ymax at Xmin (start of side 0). To do this it goes through the array of lines belonging to a particular layer. If a line is continuous, and its Xstart is equal to Xmin, its Ystart is equal to its Yend (ie it is horizontal), and its Ystart is greater than the Ystart of the lines found so far, then it is a horizontal edge line and function *SetEdgeLine* is called.

The main function of *SetEdgeLine* is to classify and store in a separate data structure for each layer the lines identified as belonging to the edge profile of the component. To do this, it operates under the command authority of *StoreEdge*, which "builds" successively the edge profile from its constituent edge lines (using a method explained later). Under particular instructions from *StoreEdge*, function *SetEdgeLine* stores the encountered edge lines either as they are, or by swapping their start and end points so that a continuous "stream" of edge lines is formed. "Stream" in this sense means that the end point of one edge line is the beginning of the next one, and so on until the end of the last edge line meets with the start of the first edge line (the horizontal line whose start is nearest to the start of side 0). **Figure 6.8** illustrates how this process changes the orientation of certain edge lines on an edge profile to form a continuous stream.

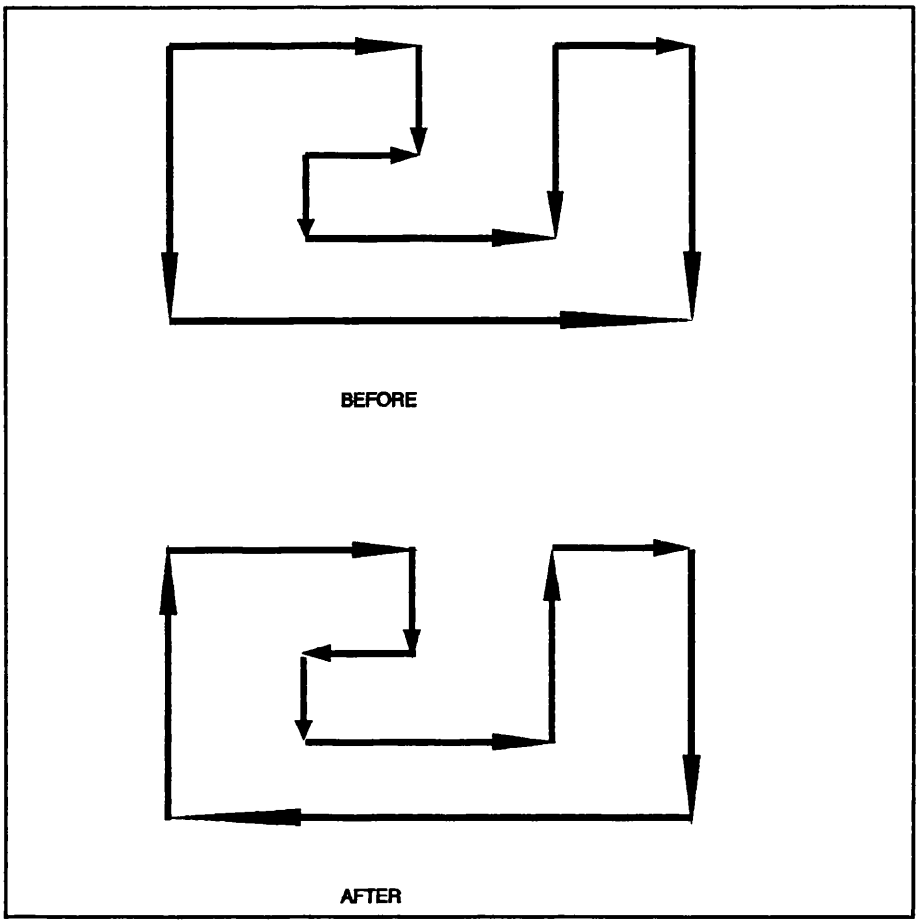


Figure 6.8 Forming a continuous "stream" of edge lines

The second task of function *SetEdgeLine* is to determine if the four corners of the edge profile are real or virtual. This is accomplished by checking if there is an edge line starting from a minimum/maximum corner point. If indeed there is such an edge line, then the corner point and the corner are declared real (true), otherwise they are declared virtual (false).

Returning to function *StoreEdge*, which has now determined the first horizontal edge line - start of side 0, and stored it using *SetEdgeLine*. The next thing *StoreEdge* does is to initialise an array called *Included*, used to ensure that any line

already determined as an edge line is not reconsidered when looking for other edge lines.

StoreEdge then moves back to the first detected edge line. Going again through all the continuous lines of the layer, it looks for a line whose start follows from the end of the first edge line (ie it looks for the next edge line of the stream). Two possibilities then arise:

1) Such a line is found. *StoreEdge* then instructs *SetEdgeLine* to store it in the edge line profile data structure as it is, and puts it in the *Included* array. It then continues its search for the next edge line of the stream.
2) A line fitting this description is not found, but instead a line whose end follows from the edge line is encountered. In this case, *StoreEdge* instructs *SetEdgeLine* to swap the line's start and end points, and then store it as an edge line. The line is also put in the *Included* array, and the search can continue for the next edge line of the stream.

The above process is repeated until the final edge line is encountered (this is the line whose end is the start of the first edge line). A component edge profile for each layer can thus be successively built and stored for further manipulation. It should be noted that function *StoreEdge* includes safety traps for reporting any errors encountered during the component's edge profile recognition.

Once the component edge profile has been determined on layer 0, function *StoreEdge* proceeds to identify the start points of the other three sides of the

component using the algorithm described previously. For this purpose it goes through the newly stored edge line data structure.

So, for finding the line whose start is nearest (or at) Ymax at Xmax (start of side 1), it searches through edge lines that have their Xstart=Xmax, and their Ystart greater than the nearest Ystart found so far. Once the search is complete, the determined edge line is marked and stored as the start of side 1 for the particular layer. Similarly, to find the start of side 2, *StoreEdge* looks for the line whose start is nearest to Ymin at Xmax. The search now is for edge lines that have their Xstart=Xmax and their Ystart less than the nearest Ystart found so far. The determined edge line is also marked and stored as the start of side 2 for that particular layer. Finally, to find the start of side 3, *StoreEdge* looks for the line whose start is nearest to Ymin at Xmin. The search this time is for edge lines that have their Xstart=Xmin and the Ystart less than the nearest Ystart found so far. The discovered edge line is marked and stored as the start of side 3 for that particular layer. The above process is repeated for all three layers of the drawing. Safety traps are again included to report any possible errors.

6.2 Identifying Multiple Through Slots

A through slot can be defined as a square groove cutting across any side of a prismatic part (**Figure 6.1**). In an engineering drawing form it can be depicted as in **Figure 6.9** below, where a simple through slot is shown on the top side of the front view of the component. The first challenge of the research work on through slot recognition was therefore how to develop an algorithm capable of exploiting the characteristics of through slot notation on an engineering drawing.

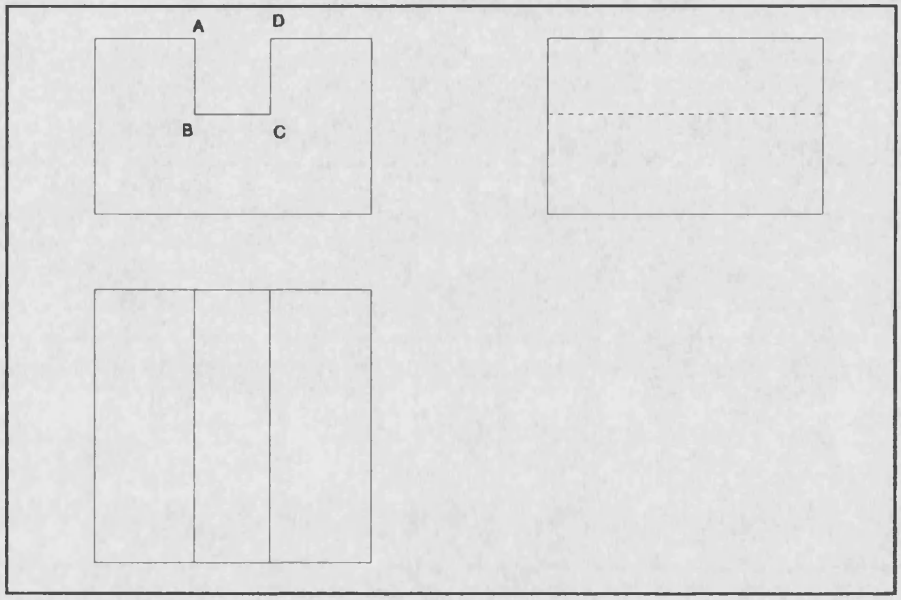


Figure 6.9 Engineering drawing notation of a simple through slot

After careful consideration, it was decided that the best way to go about recognising a through slot, would be to take advantage of the discontinuity on the edge profile, that such a slot incurred in some view of the drawing. For example, in **Figure 6.9**, such a discontinuity exists on the front view, side 0. An initial algorithm was thus developed that would be able to deal with simple through slots as follows

(refer to **Figure 6.9**):

(1) Point A is located (Xmin < X < Xmax, Ymax) where the discontinuity starts.
(2) Point B is found, and line AB confirmed as a continuous line (no secondary features) vertical to line (Xmin, Ymax)A.
(3) Point C is located, and line BC confirmed as a continuous line vertical to AB.
(4) Point D is found, and line CD confirmed as a continuous line vertical to BC and parallel to AB.
(5) Line D(Xmax, Ymax) is found and its continuity confirmed.

During its programming implementation, the algorithm was heavily modified and fine tuned. This was done in order to ensure that it would be able to cope with slightly more complicated component profiles. In particular, the algorithm found the existence of two or more simple slots on the same side of the component difficult to handle originally.

Following considerable experimentation, it was eventually decided to make use of the direction the edge lines of a profile had when a through slot existed. That is when the vital concept of sorting the edge profile in a continuous "stream" (as described in section 6.2) first emerged. Consider how the edge lines go if a simple through slot exists on side 0 of an edge profile:

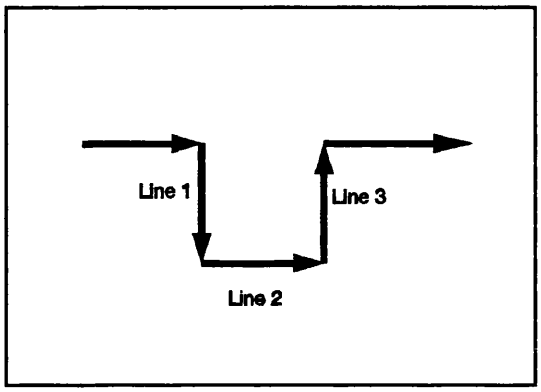


Figure 6.10 Edge line direction for a simple through slot

As it can be seen from the figure above, instead of a simple horizontal line going from left to right, the existence of the slot modifies the profile to include a top to bottom ("downwards") line (line 1), a left to right line (line 2) and a bottom to top ("upwards") line (line 3). It could therefore be surmised that if the program started going in the correct order through the edge lines of side 0 ("walking" around side 0) and it came across a "downwards" line followed by a corresponding "upwards" line, then this would indicate the presence of a slot. Furthermore, the length of the downwards reference line (line 1) would be the depth of the slot, while the distance between the reference downwards line and the corresponding upwards line (line 3) would be the width of the slot. This distance could be calculated by the difference in X coordinates of the two lines.

Once the slot and its characteristics were identified and recorded, the program could just continue walking around the edge line profile of side 0, looking for another downwards line (indicating the presence of another slot) to reference from. The process could be repeated until the end of side 0 was reached. Hence, the problem of multiple through slots on the same side of a component was resolved with

this next version of the algorithm, which could be summarised as follows:

"A through slot on side 0 is formed by any vertical upwards line to the right of a downwards reference line".

A slightly modified version of the algorithm would have to be applied to each of the other sides of the edge profile for reliable slot detection. Consider the following edge profile:

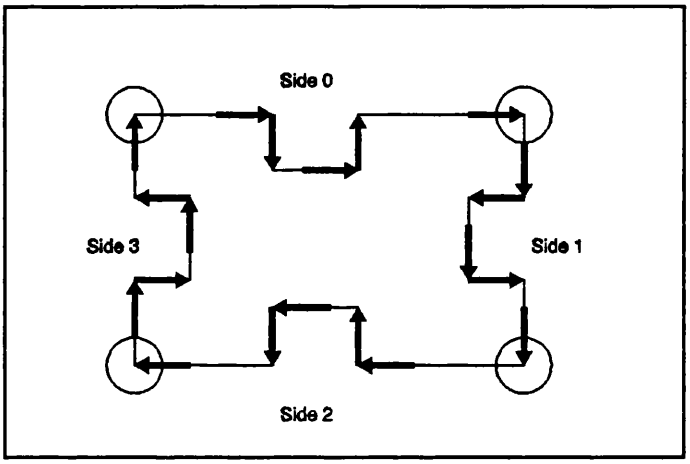


Figure 6.11 Slots in various sides of a view

It can be seen from the above figure that to detect a slot in side 1, the reference edge line would have to be a right to left line. For side 2, the reference edge line would have to be upwards (the opposite from side 0), while for side 3 it would have to be left to right (the opposite of side 1). This shows the importance of being able to distinguish where each side starts and ends, so that the different versions of the slot recognition algorithm could be applied. This requirement led to the development of the side distinction algorithm described in section 6.2.

The above concepts were all incorporated into the program code, so BUFIP could detect multiple simple through slots at any layer of a drawing by "walking" around the edge profile, and applying the slot recognition algorithms to its various sides. Once this was achieved, some more complex component geometries were then researched. In doing this it became evident that a further refinement of the algorithm was needed. Let us consider a slightly more complex through slot profile, again located at side 0 of a view:

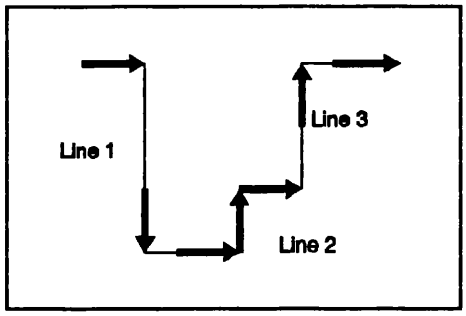


Figure 6.12 Complex through slot component profile (I)

This effectively combines two slots in one (a multiple through slot), and would produce an erroneous report if the slot recognition algorithm was applied to it, since there are now two upwards lines corresponding to the same downwards reference line. Another variation of this slot combination could be the one shown in **Figure 6.13** below:

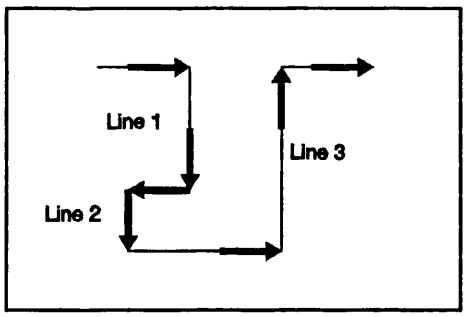


Figure 6.13 Complex through slot component profile (II)

This multiple through slot is relatively simple compared to the through slots shown in **Figures 6.14** and **6.15**:

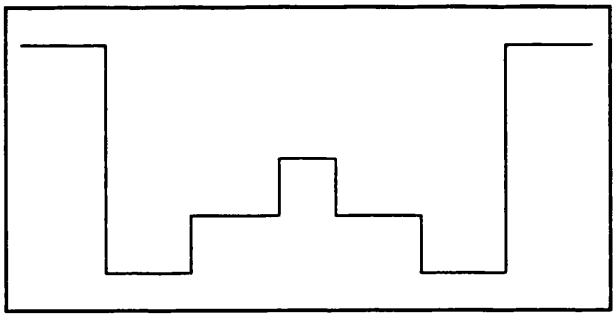


Figure 6.14 Complex through slot component profile (III)

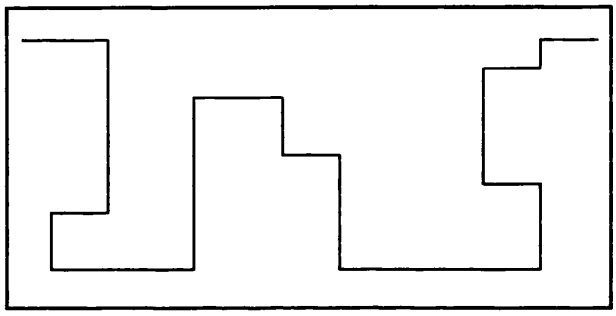


Figure 6.15 Complex through slot component profile (IV)

As even greater complexity is found when combining slots with multiple stepped faces, clearly an upgraded version of the recognition algorithm was needed, that was able to handle the highly complex profile geometries anticipated.

After much experimentation, a new upgraded version of the algorithm was eventually developed. At its very simplest it could be described as follows (for side 0):

"A through slot on side 0 is formed by any vertical upwards line to the right of a

downwards reference line, to which the reference line is exposed or partially exposed".

By "exposed" or "partially exposed" it is meant in this context that there is no visible interference between the two lines, ie there is no other line in between them. So, by applying this new rule to **Figure 6.12**, it can be observed that line 1 is partially exposed to line 2 (and thus forms a through slot), and partially exposed to line 3 (and thus forms another through slot).

Similarly, for **Figure 6.13**, line 1 is fully exposed to line 3 (one slot) and line 2 is also fully exposed to line 3 (another slot). **Figure 6.16** below shows graphically how these slots are determined for these two profiles:

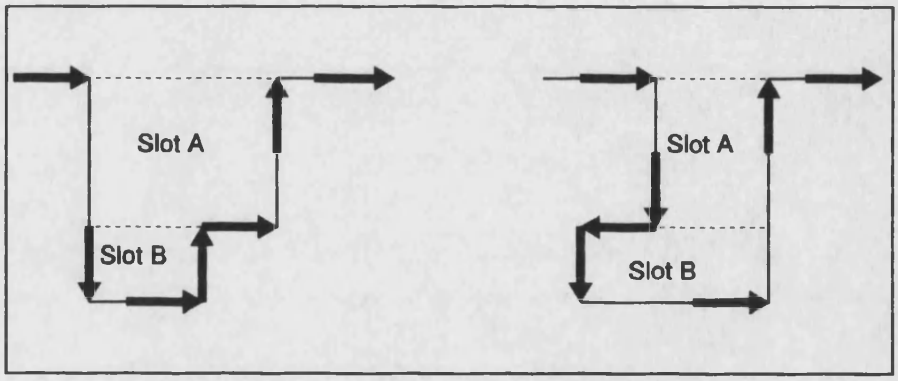


Figure 6.16 Resolved multiple through slots (I)

The width of each slot can be determined by the distance between the reference line and its corresponding upwards line, while the depth is equal to the exposed part of the reference line (this might be the whole length of the reference line, as **Figure 6.16** above shows). This new enhanced version of the algorithm now

provides effective slot detection for the complex edge profiles shown. Reconsidering **Figures 6.14** and **6.15**, they can now be easily resolved, as shown in **Figure 6.17** below:

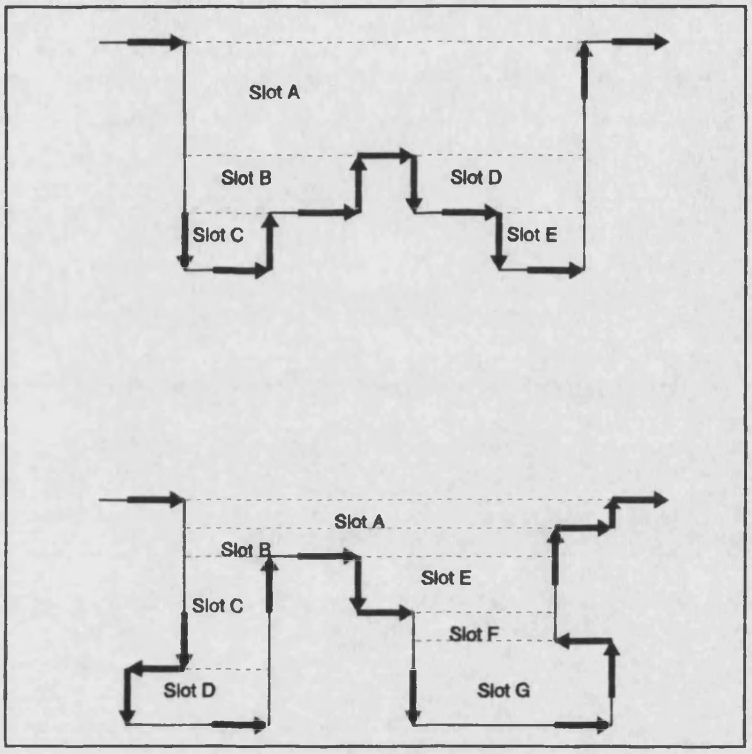


Figure 6.17 Resolved multiple through slots (II)

Again, slightly modified versions of the algorithm needed to be applied to the different sides of the edge profile to cater for the changes of direction to the reference lines (right to left for side 1, bottom to top for side 2 etc).

All these changes were effected in the next versions of BUFIP, and so the system was now able to resolve highly complex through slot geometries. However, during extensive trials of the system, another problem eventually surfaced. To illustrate it, let us consider the profile shown in **Figure 6.18** below:

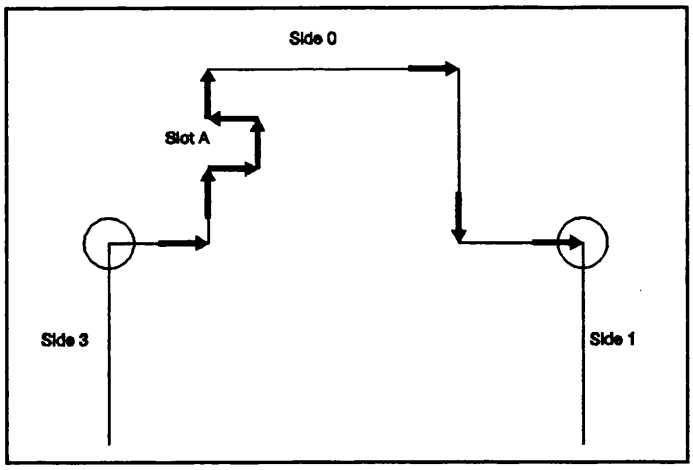


Figure 6.18 Problematic edge profile (I)

In the above profile, slot A could not be identified at all by this version of the system. This was because, according to the side classification algorithm, slot A belongs to side 0, and the slot recognition algorithm for side 0 looks for downwards reference lines. Such a line does not exist in the case of slot A, and therefore the slot is not identified. This inability of the system meant that the algorithm had to go through another major modification.

After much deliberation, it was decided to completely overhaul the slot recognition system, so that it would be able to resolve situations such as the one described above. After considerable experimentation, it was eventually decided to define slots according to the four points of the compass.

According to this convention, a North slot can be defined as a slot facing due North (**Figure 6.19**), which means that it will have a downwards reference line and an upwards corresponding line, but it could be found at any side of the component. Similarly, an East slot faces East and has a right to left reference line, a South slot

faces South and a West slot faces West. Such slots are illustrated in **Figure 6.19** below:

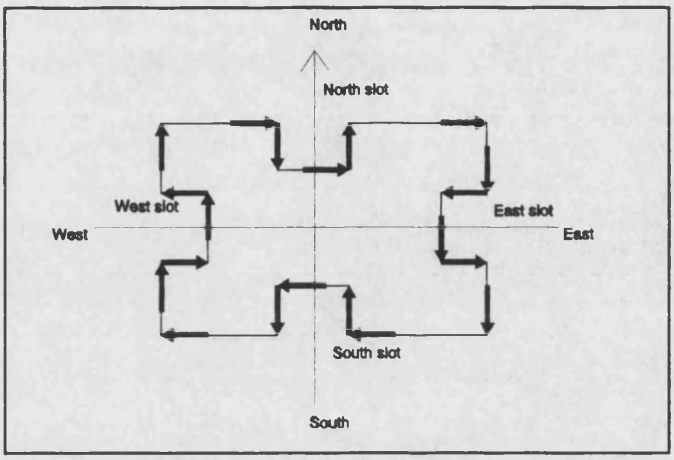


Figure 6.19 Slots defined according to the four points of the compass

Each type of slot still employs the same versions of the recognition algorithm discussed previously. For example, the North slot recognition algorithm would be

"A North through slot is formed by any vertical upwards line to the right of a downwards reference line, to which the reference line is exposed or partially exposed".

The main difference in the new implementation of the algorithm though, is that this algorithm can now be applied to any side of the component. Hence the system can walk around the complete edge profile looking for North slots, then for East slots and so on. The allocation of a discovered slot to a certain side for process planning purposes still applies however.

This new approach solves the types of problem similar to the one described

in **Figure 6.18**. For example, slot A in **Figure 6.18** can now be identified as a West slot on side 0. The new convention was incorporated into the program code and was found to successfully resolve any edge profile that it was given. This is the slot recognition method currently employed by BUFIP. However, it is not without its limitations.

It was discovered during experimentation that the system would occasionally double report the same area of an edge profile. To see how this might happen, let us consider the profile shown in **Figure 6.20** below:

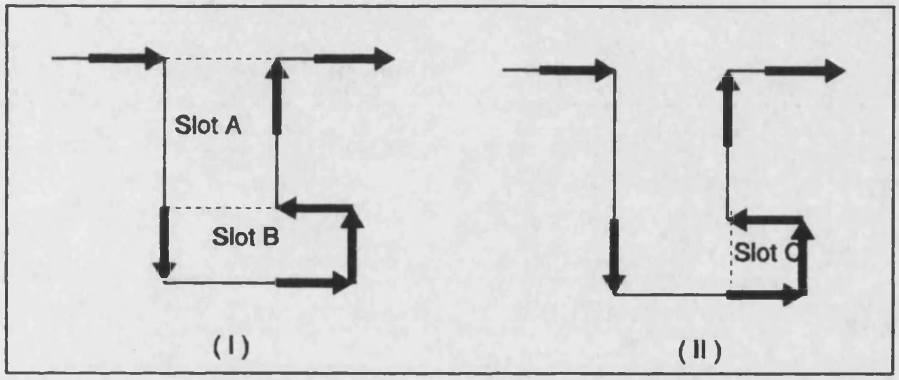


Figure 6.20 Problematic edge profile (II)

When the North slot algorithm goes through the profile, it will pick up two North slots, A and B, as shown in **Figure 6.20 (I)**. However, when the West slot algorithm is applied later on to the same profile, a West slot (C) will also be picked up (**Figure 6.20 (II)**). Hence, a certain area of the profile (the area covered by slot C) will be reported twice.

This type of situation has not been resolved as yet, and is considered to be an

area of future research. None the less, it was felt that it was better to have the system, occasionally double report the same area, than to completely miss a slot (as in **Figure 6.18**).

The following describes how the algorithms for slot recognition have been iincorporated into the system. The task of identifying through slots is handled completely by the **slots.c** program. This consists of the following defined functions:

Throughslot: The main part of the program. Responsible for activating the other slot recognition functions for every side of every layer of the drawing.

AtExtreme: Calculates when the edge line profile has reached the end (extreme) of a certain side.

InitialiseSlot: Initialises (sets to zero for a start) the through slot data structure.

RecordSlot: Records and stores the various slot parameters (width, depth, distance from edge).

ReportSlot: Reports to the output file the discovered slot type, its location, and its associated parameters.

NorthSlot: Applies the recognition algorithm for North facing slots by walking around a side of the edge profile. Also activates functions InitialiseSlot, AtExtreme, RecordSlot, ReportSlot.

EastSlot: Applies the recognition algorithm for East facing slots by walking around a side of the edge profile. Also activates functions InitialiseSlot, AtExtreme, RecordSlot, ReportSlot.

SouthSlot: Applies the recognition algorithm for South facing slots by walking around a side of the edge profile. Also activates functions InitialiseSlot, AtExtreme, RecordSlot, ReportSlot.

WestSlot: Applies the recognition algorithm for West facing slots by walking around a side of the edge profile. Also activates functions InitialiseSlot, AtExtreme, RecordSlot, ReportSlot.

A flowchart of the general operation sequence of the program can be seen in **Figure 6.21**.

The program starts by executing function *Throughslot*, which in turn executes in succession the functions *NorthSlot, EastSlot, SouthSlot* and *WestSlot* for each and every side of every view of a prismatic part drawing. These four functions, each one responsible for identifying and reporting any slots facing a respective point of the compass (North, East, South, West) perform the main processing required for correct slot extraction.

Each of these four functions includes an appropriate form of the slot recognition algorithm described earlier, plus execution calls for functions *AtExtreme, InitialiseSlot, RecordSlot* and *ReportSlot* so that the identified slot(s) can be properly recorded and reported. The following describes in more detail how the first of these four functions, *NorthSlot* operates.

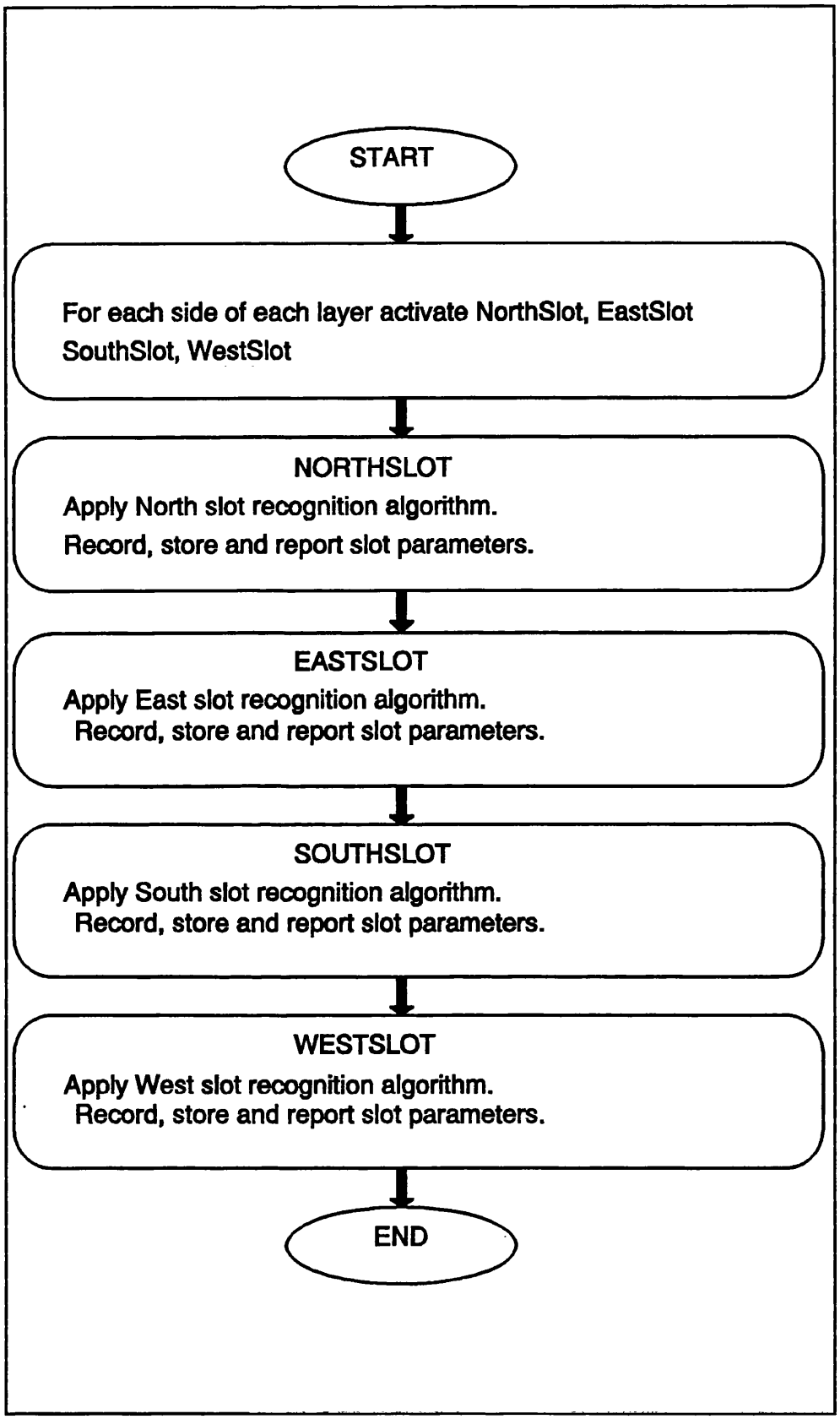


Figure 6.21 Operation sequence flowchart for function *Throughslot*

Function *NorthSlot* starts by walking around a side of an edge line profile (as indicated to it by *ThroughSlot*) from the side's start point to its end point. It looks for reference lines heading downwards, that is the Ystart of an edge line must be greater than its Yend. Once such a line is found, it activates function *InitialiseSlot* to initiate the slot data structure. It then continues walking around the edge profile looking for a parallel upwards line to the right of the reference line. Such a line would have its Xstart greater than the Xstart of the reference line, and its Ystart different than its Yend (ie it will not be horizontal).

Once such a parallel line is found, it is compared with the reference line in order to establish if the reference line is exposed or partly exposed to it. For this purpose four variables, RStart (Reference Start), REnd (Reference End), PStart (Parallel Start) and PEnd (Parallel End) are used for comparing the Y coordinates (in this case) of the two lines. By comparing where PStart and PEnd lie with regards to RStart and REnd, the exposed length of the reference line can be determined.

Once this exposed length is established, function *RecordSlot* is activated. This stores in the slot data structure the characteristics of the currently identified slot (width, distance from edge etc). Function *ReportSlot* is next activated for reporting and recording in the output file the type of slot encountered (in this case North), its location (side number and layer name) and its various parameters. In particular, the output format of the system for each detected slot is as follows:

"(Type) slot found on layer (layername) Side (sidenumber):

X=(Reference line Xstart) Y=(Reference line Ystart)

Distance from Edge=(distance), Width=(width), Depth=(depth)"

where

(Type)=the type of detected slot (North, East, South, West)

(layername)=the layer that the slot belongs to

(sidenumber)=the side that a slot belongs to

X=X coordinate that the slot starts from (ie the Xstart of the reference line)

Y=Y coordinate that the slot starts from (ie the Ystart of the reference line)

(distance)=the distance from the bottom of the slot to the edge line profile; this might or might not be equal to its depth (in mm)

(width)=the width of the slot (in mm)

(depth)=the actual depth of the slot (in mm).

The slot characteristics reported by the system are illustrated in a graphical form for a simple North slot in **Figure 6.22** below:

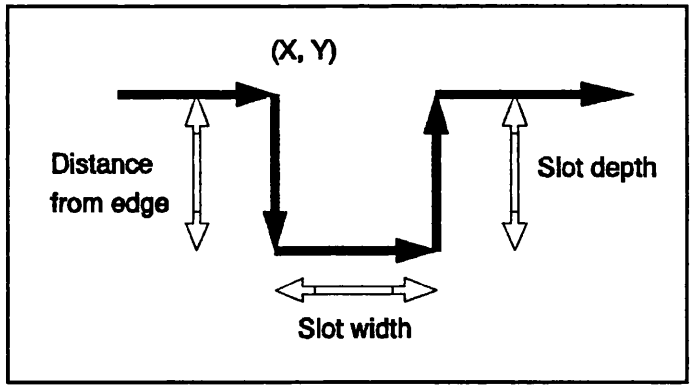


Figure 6.22 Characteristic parameters of a simple North slot

Once a slot is reported, function *NorthSlot* proceeds looking for another upward parallel line(s) further to the right to "cover" the rest of the exposed length

of the reference line (if applicable), and when found the new slot(s) is again reported. Otherwise, it continues walking around the side, looking for a new downwards reference line. The whole process is repeated for such a new reference line, and so on until function *AtExtreme* (employed by *NorthSlot*) detects that the end of a side has been reached.

Function *NorthSlot* then terminates, and passes control back to the main part of the program, *Throughslot*. A flowchart of the general operation sequence of function *NorthSlot* can be seen in **Figure 6.23**. *Throughslot* now activates the next slot recognition function, *EastSlot*, to be used for the same side of the edge profile.

Function *EastSlot* (as well as *SouthSlot* and *WestSlot*) is similar in its operation sequence to *NorthSlot*, and uses the same support functions (*AtExtreme, InitialiseSlot, RecordSlot* and *ReportSlot*). Its main difference is that it uses a modified form of the slot recognition algorithm (reference lines are now from right to left), and as a consequence, slightly different computations for determining RStart, REnd, PStart, PEnd and therefore the exposed length of the reference line. Functions *SouthSlot* and *WestSlot* also use correspondingly modified forms of the algorithm with similar differences in computations.

Once all four slot recognition functions are executed for the same side of an edge profile, *Throughslot* moves to the next side, and so on until the complete edge profile for a layer is covered. *Throughslot* then moves to the next layer of the drawing, and repeats the same process, until all slots are identified in all layers.

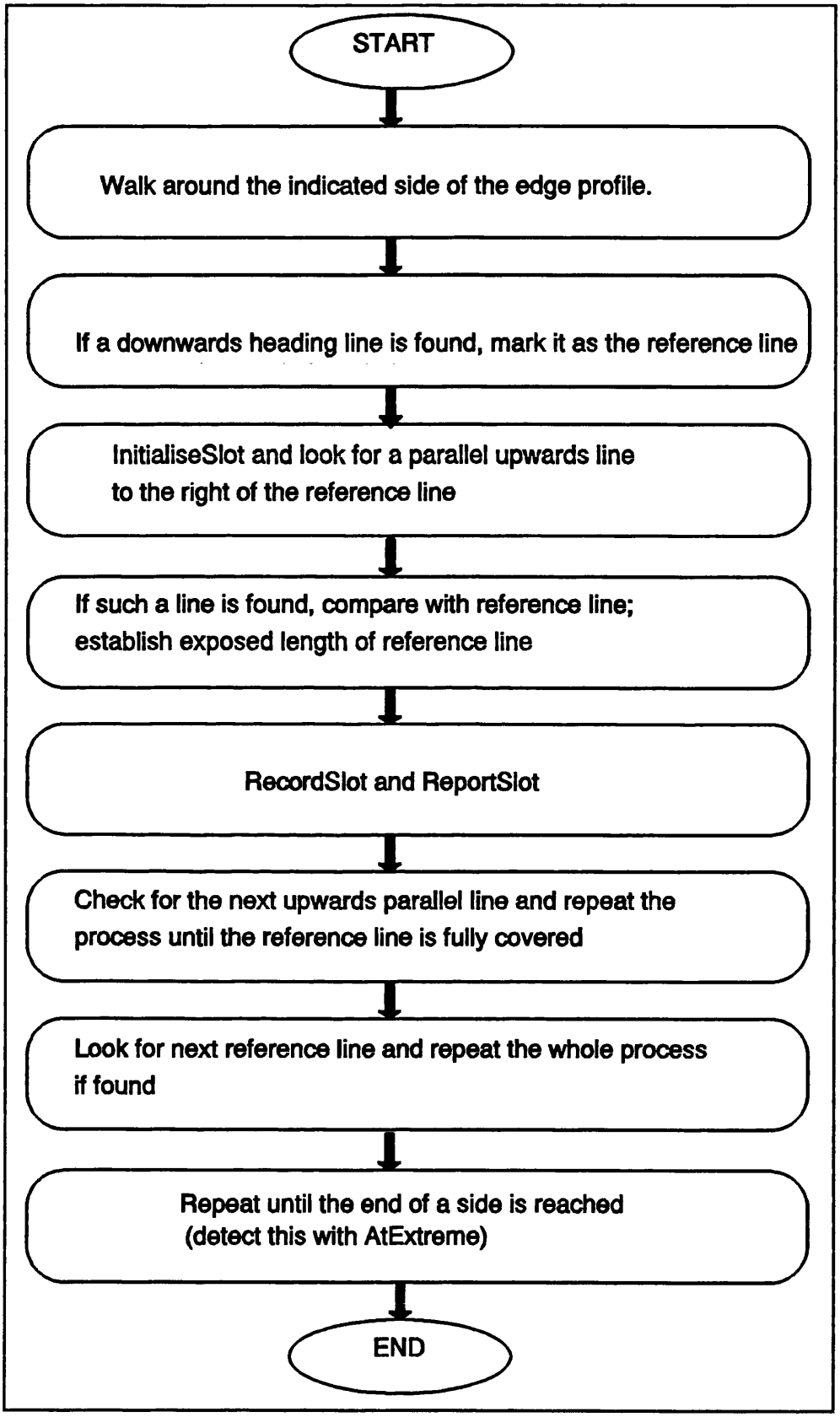


Figure 6.23 Operation sequence flowchart for function *NorthSlot*

6.3 Identifying Multiple Stepped Faces

A stepped face, as the name suggests, is simply a face that starts and ends at different levels of a prismatic part (**Figure 6.1**). Given the constraint of only allowing horizontal or vertical surfaces, this has the effect of producing a "step" on that surface that joins the two levels together. In an engineering drawing form (**Figure 6.24**) a simple step is shown at the top left hand corner of the component (corner 0) of the front view. As with through slots, a way had to be found to take advantage of the characteristics of stepped face notation on an engineering drawing.

Following some experimentation, it was decided , in a similar manner to through slots, to exploit the change in the edge profile that a step incurs in a corner of a drawing view. For example, in **Figure 6.24** such a change exists in corner 0 of the front view. The following algorithm was then developed, able to deal with such simple stepped faces:

" A step can be found at any corner of the drawing of a component which does not have an edge line starting from or ending at it (ie if the corner is not real but "virtual")".

In other words, if one determines the minimum and maximum coordinates forming the four corner points of a prismatic part on each layer of the drawing, and then checks if there are any lines starting from or ending at these points, one can detect the existence of a stepped face.

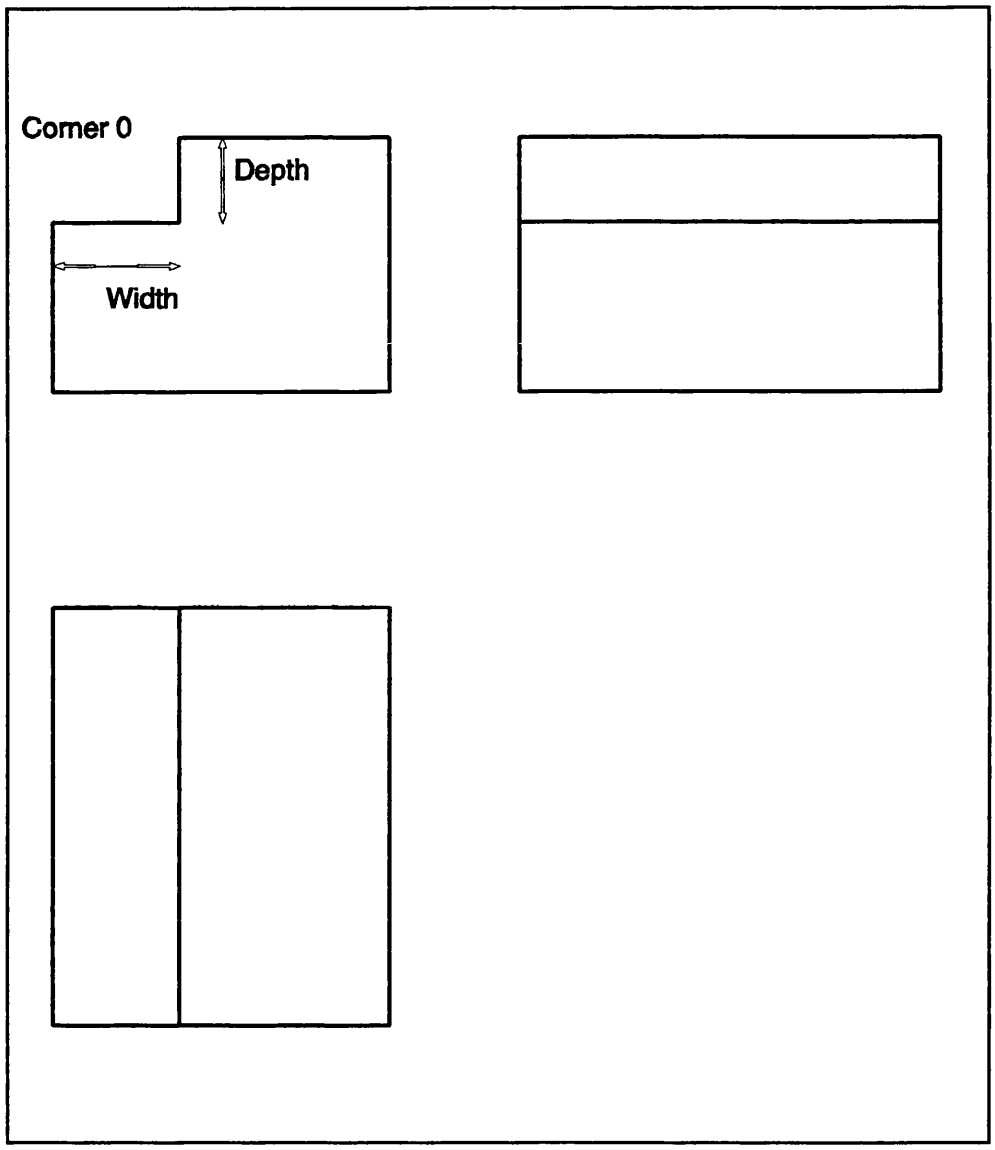


Figure 6.24 Step notation on engineering drawings

This basic algorithm is still currently used by BUFIP. Function *SetEdgeLine* from **props.c** (described in detail in section 6.2) applies this algorithm to the four corners of each view of the drawing to determine the existence of a step (or steps).

Once the existence of a stepped face is discovered and located, then its characteristics have to be extracted. These are the step's width and its depth (**Figure**

6.24). In a manner similar to through slots, the step's width can be defined as its dimension in the horizontal plane (X-axis), while the step's depth is its dimension in the vertical plane (Y-axis).

Instead of developing an algorithm to determine just these simple step characteristics, it was decided from the outset to develop one that would cater for the existence of multiple steps on the same corner of a layer.

After considerable experimentation, an algorithm was eventually developed and validated. To analyze in detail how it works, let's consider how the edge profile of corner 0 at a layer is modified by the existence of one simple (**Figure 6.25 I**) or three multiple steps (**Figure 6.25 II**):

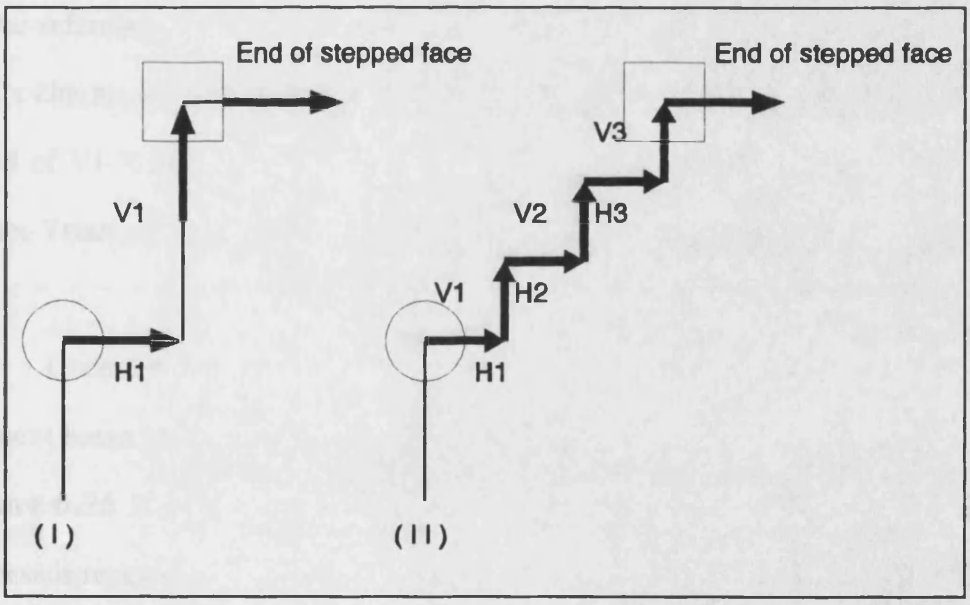


Figure 6.25 Stepped edge profile on corner 0

In both cases, a common reference point has to be found from which to start

determining the step's characteristics. This was decided to be the starting point of a side, side 0 in this case, (shown encircled in **Figure 6.25**), determined earlier using the side distinction algorithm, described in section 6.2.

Beginning therefore from the starting point of side 0, and going through the edge line data structure until a line with Yend=Ymax is found (indicating the end of the stepped face, and shown in **Figure 6.25** within a square box), the system looks for horizontal lines. When such a line is found (line Horizontal 1, H1, in **Figure 6.25**), the program flags it as a reference line and marks it.

It then continues moving around the object, looking for the next vertical ascending line in order to establish the step's width. For reasons of allowing for slot interference (explained later), such a line would have its Yend larger than the Yend of the reference line. In **Figure 6.25**, line V1 (Vertical 1) is thus found. Now the step's characteristics can be easily determined. Its width will be equal to:
Xend of V1-Xstart of the reference line, while its depth will be
Ymax-Ystart of the reference line.

Once the step characteristics are extracted, the system then proceeds to find the next horizontal line, which will form the basis of the following step. Line H2 in **Figure 6.25 II** is thus found and marked as the new reference line, and the above process is repeated for determining the new step's characteristics. This procedure can be repeated as many times as required, until all the steps for corner 0 have been identified (ie the vertical line with Yend=Ymax is encountered). The system is thus able to deal with any number of multiple steps on a corner of a view. When corner

0 is completed, the program can move on to the next corner of the drawing (corner 1).

In a manner reminiscent of the through slot recognition algorithms, the step algorithm also needs modifications for coping with other corners of a view. This time though, the changes required are quite major. Consider what happens to corner 1 of a view when, multiple steps are located there (**Figure 6.26**):

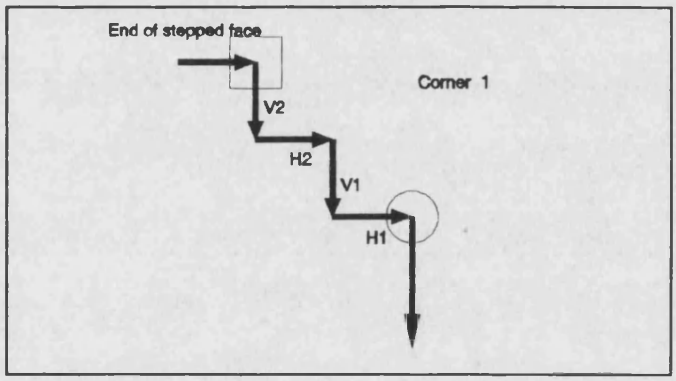


Figure 6.26 Stepped edge profile for corner 1

As it can be seen from **Figure 6.26**, the initial circled common reference point for the search (start of side 1) now lies below the edge lines forming the steps. This means that the program will have to "walk" backwards to find any reference horizontal lines forming steps. In other words, it will have to decrement through the edge line data structure until it encounters a line with Ystart=Ymax (indicating the end of the stepped face).

Also, when a horizontal reference line is found (say H1 in this case) the system needs to travel backwards, looking for the previous vertical descending line

to establish the step's width. Such a line will now have its Ystart larger than the Ystart of the reference line. When such a line is found (V1 in this case), the step's width will then be equal to Xend of the reference line-Xstart of V1, but its depth will still be Ymax-Ystart of the reference line.

If corner 2 of a layer includes multiple steps (**Figure 6.27**) the algorithm would have to increment (walk forwards) again from the initial common reference point (start of side 2). However, the end of the stepped face in this case is met with the line that has Yend=Ymin.

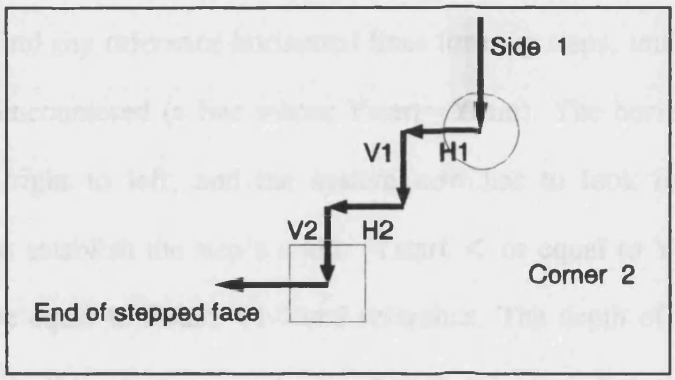


Figure 6.27 Stepped edge profile for corner 2

Also, the horizontal reference lines are now right to left (ie their Xstart will be larger than their Xend), but the system should still look for the next vertical descending line to establish the step's width (Yend < Yend reference). This will now be equal to Xstart reference-Xend of V1. The depth of the step is also calculated differently. It will now be equal to Ystart reference-Ymin.

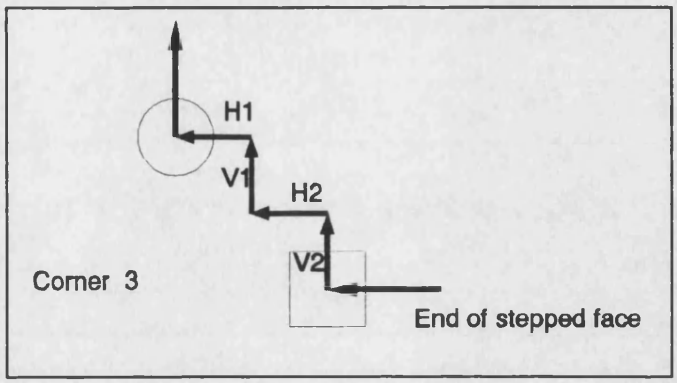


Figure 6.28 Stepped edge profile for corner 3

Finally, for corner 3 of a layer that includes multiple steps it can be seen from **Figure 6.28**, that the algorithm would have to walk backwards again (decrement) to find any reference horizontal lines forming steps, until the end of the stepped face is encountered (a line whose Ystart=Ymin). The horizontal reference lines are again right to left, and the system now has to look for the previous ascending line to establish the step's width (Ystart < or equal to Yend reference). This will now be equal to Xstart V1-Xend reference. The depth of the step can be calculated, as in the previous case, by Ystart reference-Ymin. **Figure 6.29** summarises how the algorithm works for all four corners of a drawing's view. The figure indicates the starting points of the various sides with circles, and the ending points of the stepped faces with squares.

Once the methodology of the recognition algorithm was resolved, developed in code, tested and made to work for multiple steps for all four corners of a layer, it was then further enhanced to cater for the combined existence of multiple slots. Reconsidering the case of a step on corner 0, but this time combined with a through slot on the horizontal (**Figure 6.30**).

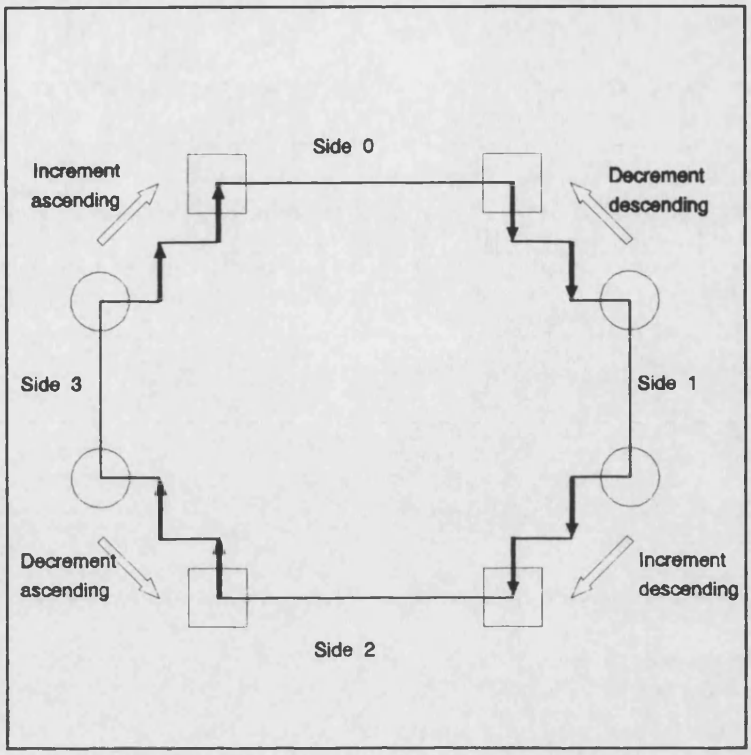


Figure 6.29 Direction of search for steps on the four corners of a view

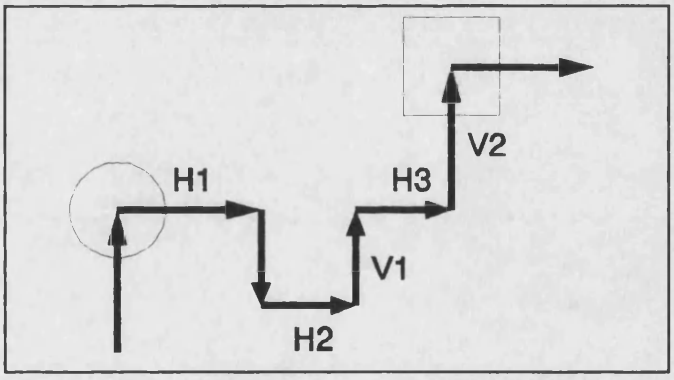


Figure 6.30 Step combined with slot on the horizontal (I)

The challenge was how to make the system ignore the through slot lines, and effectively bypass them when determining the step's characteristics. As using the step recognition algorithm described above, line H2 would mistakenly form the basis of

a step, with line V1 determining its potential width. Also, the width of the step formed by H1 would be mistakenly reported as Xend V1-Xstart reference, since V1 is the next vertical ascending line. The problem was therefore two-pronged: it had to be solved in the horizontal plane (ignoring false horizontal reference lines) and in the vertical plane (ignoring false vertical ascending lines).

To resolve the problem in the horizontal plane, a new variable, ExtremeY, was introduced. This takes the Y value of the start of the current horizontal reference line, and is initially set to Ymin for corner 0. When the first horizontal reference line H1 is encountered, ExtremeY becomes equal to Ystart H1. The program can now be instructed to ignore any subsequent horizontal lines whose Ystart is equal or less than the current ExtremeY. Hence, in the example of **Figure 6.30**, lines H2 and H3 will be ignored. If however a valid (Ystart > ExtremeY) horizontal line is encountered, it becomes the next reference line for step recognition, and ExtremeY is now set to the Ystart of the new reference line. The above process can now be repeated until the end of the stepped face. In other words, ExtremeY literally "raises the stakes" for the horizontal lines to pass before qualifying as step reference lines. The same algorithm can be applied to corner 1. For corners 2 and 3 however, ExtremeY starts from the maximum value Ymax, and eventually descends to Ymin with each new approved horizontal reference line.

The vertical plane problematic situation was resolved by stipulating that the next vertical ascending line for determining a step's width for corner 0 should have its Yend > Yend reference. This effectively excludes line V1 in **Figure 6.30**, and correctly selects line V2 for the width calculation. This condition also takes care of

situations similar to **Figure 6.31** below:

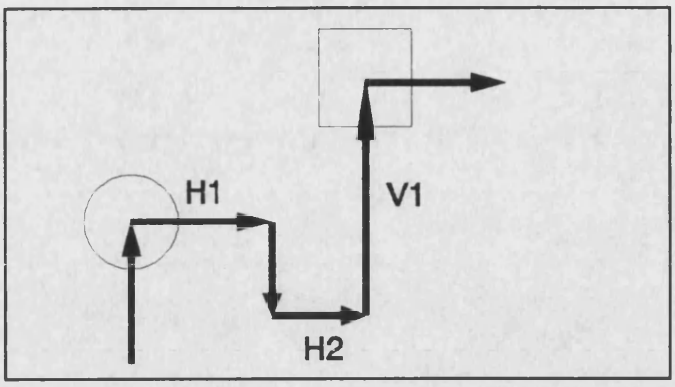


Figure 6.31 Step combined with slot on the horizontal (II)

In this case, despite the slot being combined with V1, line V1 is again properly selected for the step width calculation. The solution is applied in appropriately different forms (for ascending or descending lines) to the other three corners of a view, as mentioned previously in the discussion. Both of the above discussed approaches are also able to handle multiple through slots combined with multiple steps in similar situations without any modification. The operation of ExtremeY and the vertical line conditions are unaffected whatever the number of slots and steps.

Considering corner 0 of a view again, if the slot(s) are located in the vertical, as in **Figure 6.32** below, and despite the previously mentioned precautions put in place, the step's width would be mistakenly reported as Xend V1 - Xstart reference. To rectify this situation another condition was implemented. For corner 0 this took the form of:

"any vertical ascending line above and to the left of the selected ascending line reduces (partly or fully) the potential step's width".

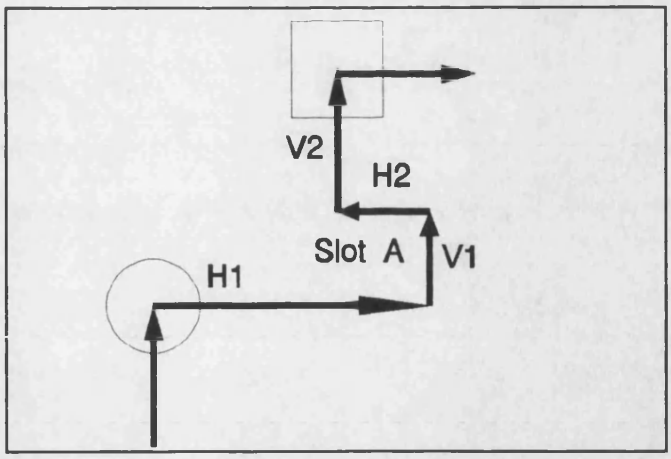


Figure 6.32 Step combined with a slot on the vertical (I)

In other words, the system, when having correctly selected a vertical ascending line for determining the step's width (V1 in this case) should further check for other vertical ascending lines "overhanging" the current line. For corner 0 these would be lines whose Ystart < Yend and whose Yend > Yend reference and whose Xstart < Xstart of the currently selected ascending line. Line V2 is thus picked up, and the new width of the step will be equal to Xend of new ascending line-Xstart reference (ie in this case Xend V2-Xstart reference). If another line further overhang V2, then this would have formed the basis for calculating the step's width, and so on. This approach also resolves the type of situation shown in **Figure 6.33** below:

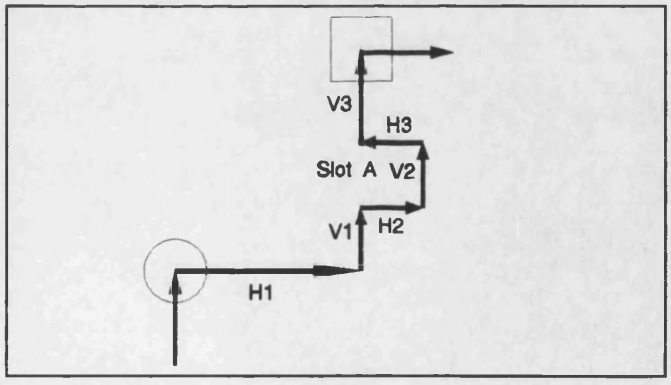


Figure 6.33 Step combined with a slot on the vertical (II)

In this case, lines H2 and V2 could be reported as a potential step. However, since line V3 overhangs V2, the step's new width would be Xend V3-Xstart H2=0 (line V3 fully covers the potential step's width), and the potential step would not be reported at all. This is correct, since lines H2, V2 and H3 form in fact a through slot.

The above restriction on determining a potential step's width is again applied with appropriate modifications to the other three corners of a layer. Similar to the other adopted solutions, this strategy can cope with multiple through slots combined with multiple steps in similar situations without any required changes. The system simply "shrinks" the step's width with each newly discovered overhanging vertical line until the end of the stepped face is encountered.

The following describes how all of these concepts are applied in the step recognition program, **steps.c**. **Steps.c** consists of a single function, *Step*, which is responsible for the identification and classification of stepped faces at all corners of a layer.

Function *Step* is divided in four distinct parts, one for each corner of a layer, and is executed for all corners of the three main layers of a drawing. Each part contains the respective version of the recognition algorithm discussed previously, along with the appropriate precautions for coping with the existence of combined multiple slots.

For example, *Step* deals with corner 0 of **Figure 6.32** in the following way.

The section is activated only if there is a "virtual" corner at corner 0, indicating the presence of a step (or steps). The existence of a "virtual" corner is already determined by function *SetEdgeLine* from **props.c** (as discussed in section 6.2).

Assuming that there is a "virtual" corner at corner 0, the section starts by setting ExtremeY to Ymin. Next, commencing from the starting edge line of side 0 (determined using function *StoreEdge* from **props.c**, as discussed in section 6.2) the program increments through the edge line profile until it reaches the edge line whose Yend=Ymax. All this time it looks for horizontal lines (Xstart < Xend) whose Ystart is larger than the current ExtremeY (Ymin).

Once such a horizontal line is found, it is marked as the current reference line, and ExtremeY is set to the Ystart of this line. The program then continues moving in an incremental order, looking for the next upward line (whose Yend, as mentioned previously, must be larger than the Yend of the reference line) to establish the step's maximum potential width. When a line meeting this criterion is discovered, its Xstart is assigned to a variable called StepX.

Now the program implements the clause discussed previously about reducing the potential step's width if an "overhanging" vertical line is found. If such a line is discovered, variable StepX is assigned the Xstart of this line.

Finally, the program processes and records its findings about corner 0 in the output data file. This report has the following format for each encountered step:

"Step found on Layer (layername) Corner 0:

X=(StepX), Width=(width), Depth=(depth)"

where

(layernumber)=the layer code number (0 for front view etc)

(StepX)=the Xstart of the ascending line for this step; this provides accurate location of the current step when multiple steps are involved.

(width)=the step's width. This is equal to StepX-Xstart reference (in mm).

(depth)=the step's depth. This is equal to Ymax-Ystart reference (in mm).

A flowchart of the operational sequence of this section of function *Step* can be seen in **Figure 6.34**. Once the current step is reported, the section looks for the next horizontal line (if applicable) to be marked for reference (its Ystart should be larger than the current ExtremeY). ExtremeY is then set to Ystart of the new reference line, and the identification process is repeated for the new step. This procedure is followed until the final vertical edge line for corner 0 is encountered (its Yend=Ymax), signalling that all steps for corner 0 have been processed.

Function *Step* then continues with the next part of the program, responsible for reporting steps at corner 1. The parts for corners 2 and 3 are next executed, until all steps in all four corners of a layer are extracted. The process is repeated for the remaining views of a drawing until all stepped faces have been identified, whereupon the program ends execution. The program has been tested on a significant variety of sample component stepped profiles, and was found capable of resolving any combination of multiple slots with multiple stepped faces it was given.

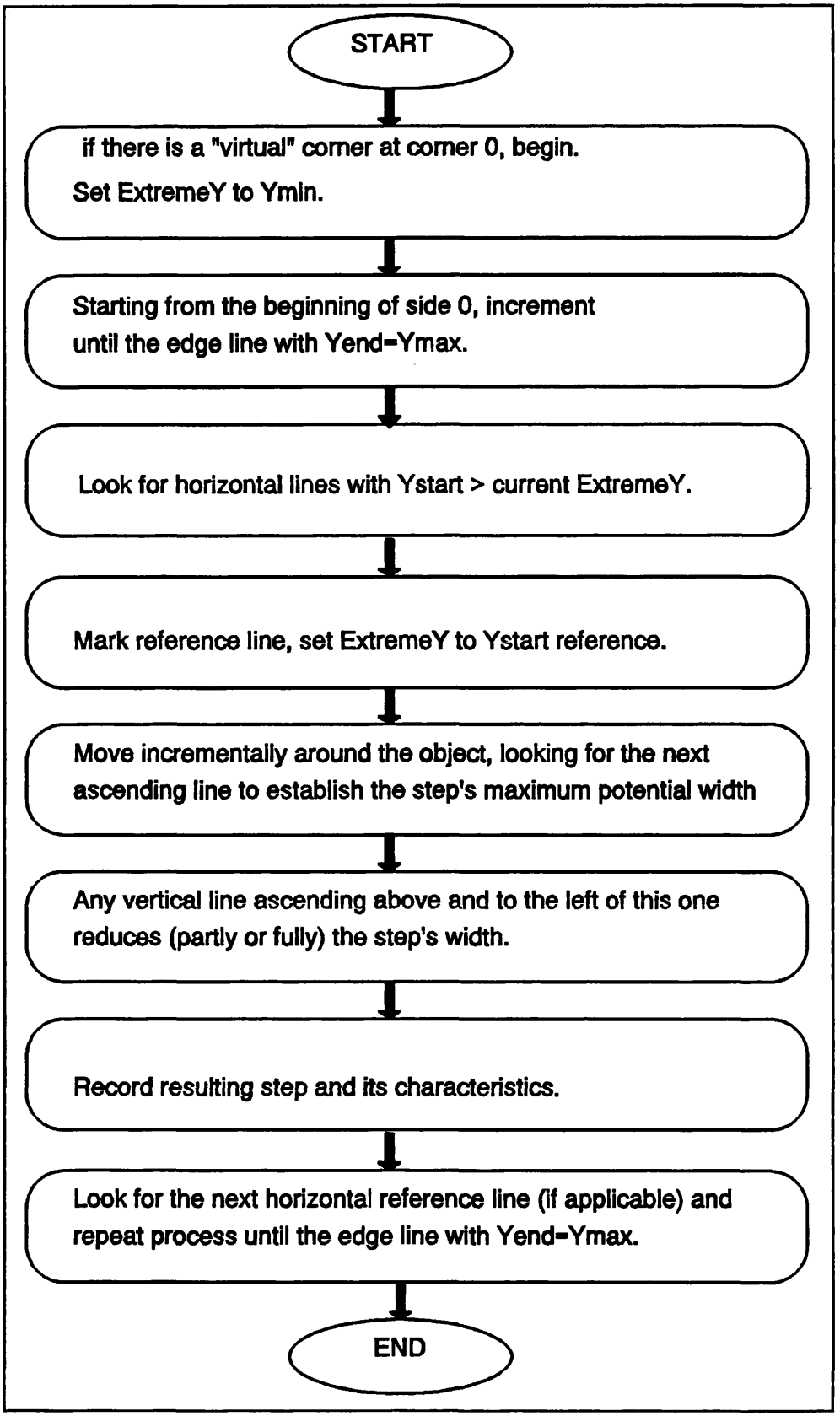


Figure 6.34 Operation sequence flowchart for the corner 0 part of **steps.c**

6.5 Identifying Pockets

6.5.1 Introduction

Pockets, in this context, are defined as the machined depressions on certain surfaces of a component. They can be generally classified into three main categories: open, side, and closed (**Figure 6.1**). Open pockets are located at the corners of a component, side pockets along the sides, and closed pockets are contained within the part's surface area. In an engineering drawing form, these three different kinds of pockets can be depicted as in **Figure 6.35** below, where, in this case, they can be found at the front view of the drawing.

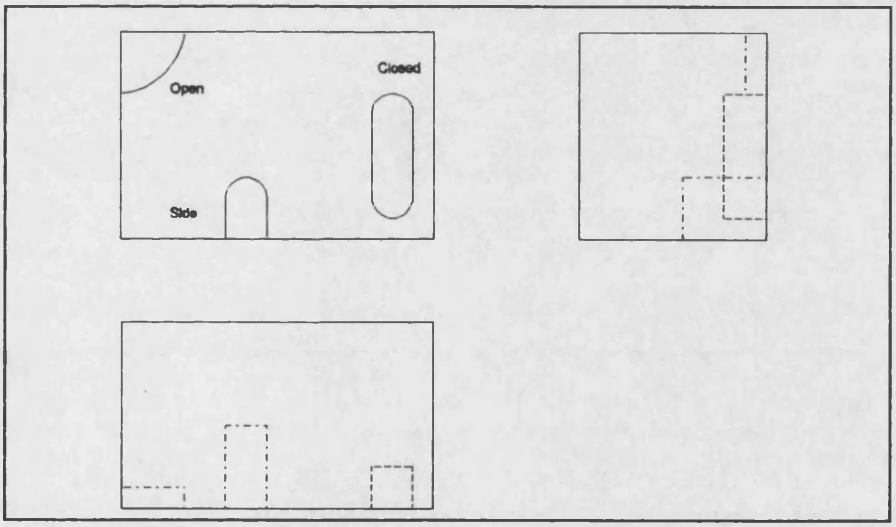


Figure 6.35 The three different types of pockets

Before setting out to develop a methodology for identifying pockets, it was decided from the outset that because of the research nature of the work and due to time limitations, pocket interaction would not be considered. This meant that pockets

would not be allowed to overlap or combine with each other (for example two combined closed pockets, one within the other). Also, only "orthogonal" pockets would be researched, that is pockets having a horizontal or vertical orientation only (ie not "inclined" relative to an edge, or of non-homogeneous depth).

Taking these limitations into account, a strategy for identifying pockets was developed. A crucial element of this strategy (explained in section 6.5.3) was the ability to determine the exact coordinates of the start and end points of the arcs that form the corners of the pockets. These coordinates were not readily provided by the AUTOCAD database, so an algorithm able to extract them from the available information had to be incorporated.

6.5.2 Determining an arc's start and end points coordinates

By default, AUTOCAD describes arc entities in an anti-clockwise direction. An angle measurement convention is employed, whereby angles are measured anti-clockwise relative to a 0 deg axis being equivalent to the horizontal plane X axis for normal coordinates. Hence, using this measurement system, 90 deg would be equivalent to the vertical plane Y axis, 180 deg to the -X axis, and 270 deg to the -Y axis. **Figure 6.36** graphically illustrates this convention for a 90 deg arc (an arc having a contained angle of 90 degs). The centre of the arc is shown encircled, while its start and end points are highlighted within square boxes:

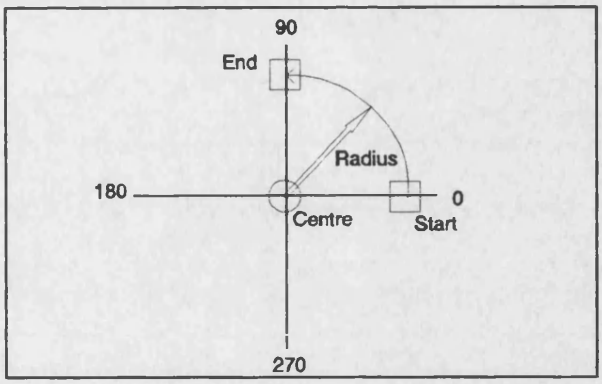


Figure 6.36 AUTOCAD's measurement convention for arcs

The coordinates of the centre of the arc (Xcentre, Ycentre) are provided by AUTOCAD, along with its radius of curvature (radius R), the distance between the arc and its centre. The start and end angles of the arc are also given, measured anti-clockwise from AUTOCAD's relative zero, mentioned previously. The start angle, as the name implies, is the point where the arc starts (0 deg in **Figure 6.36**), while the end angle is the point where the arc ends (90 deg in **Figure 6.36**).

All of these parameters are extracted and interpreted by BUCADIP, as discussed in **Chapter 4**. The challenge was to develop an algorithm able to determine the X and Y coordinates of the start and end points of an arc by manipulating the provided parameters.

Since the system deals with "orthogonal" pockets only, the required algorithm was soon realised. This is based on the fact that all the arcs that form orthogonal pockets start from or end at only four very specific angles: 0, 90, 180 or 270 degs. These coincide with the normal coordinates X and Y plane axes. In **Figure 6.36**, it can be seen that the orthogonal 90 deg arc starts at 0 deg (X-axis) and finishes at 90

deg (Y-axis). Thus, for this arc, the start and end point coordinates will therefore be:

Xstart = Xcentre + Radius

Ystart = Ycentre

Xend = Xcentre

Yend = Ycentre + Radius

If the arc started at, say, 180 deg and ended at 270 deg (**Figure 6.37**), then its start and end point coordinates would be

Xstart = Xcentre - Radius

Ystart = Ycentre

Xend = Xcentre

Yend = Ycentre - Radius

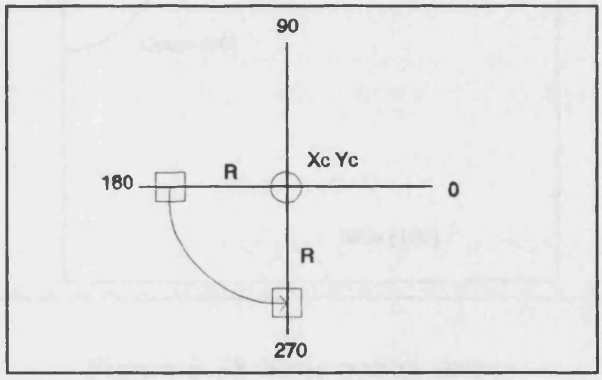


Figure 6.37 Start and end points for a 180/270 arc

As it can be seen from the previous examples, the coordinates of the start and end points of any orthogonal arc can hence be found by adding or subtracting the radius of curvature to or from the coordinates of the centre, depending on where the arc starts and ends. This algorithm can also be readily applied in programming code,

and enables the identification of the actual pockets themselves.

6.5.3 The strategy for pocket identification

In order to identify pockets, it was decided from the outset to take advantage of their most characteristic object in engineering drawing notation: the existence of arcs. These would have a contained angle of 90 or 180 degs, if they were to represent the orthogonal pockets under study. Additionally, for the preliminary basic pocket shapes examined, 90 deg arcs indicated the presence of open pockets, while 180 deg arcs indicated side pockets, as **Figure 6.38** below shows:

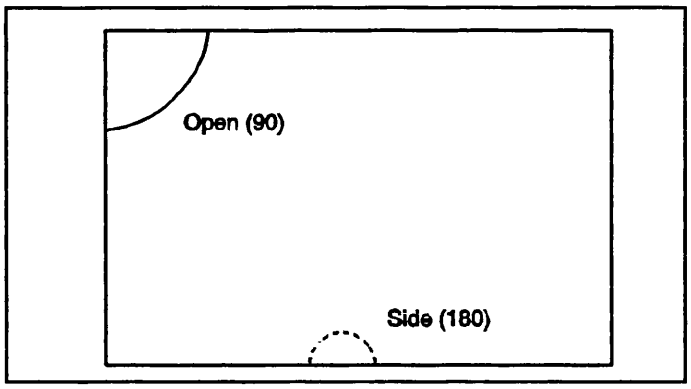


Figure 6.38 Basic pocket shapes

It should be noted at this point that this method will not cater for pockets having their centreline outside the edges of the component. This is because their constituent arcs would not have a contained angle of 90 or 180 degs, and hence the resultant pockets would not be orthogonal. However, this limitation could be resolved in a future version of the system, as indicated in section 9.2.

Figure 6.38 also demonstrates another pocket attribute that could be identified at this stage: if the arc was DASHED or HIDDEN (a property interpreted by BUCADIP), then this implied that the pocket would also be "hidden" (ie originating from the opposite side of the component depicted on that particular view of the drawing). The location of the identified pocket(s) on any view could be pinpointed by the coordinates of the arcs' centre(s), while their dimensions (length, width) could be determined by their radius of curvature R. This basic algorithm was applied and tested in programming code, and was found to work successfully. However, it was soon superseded by the requirement for resolving more complex pocket profiles and attributes. Consider for example the pockets shown in **Figure 6.39** below:

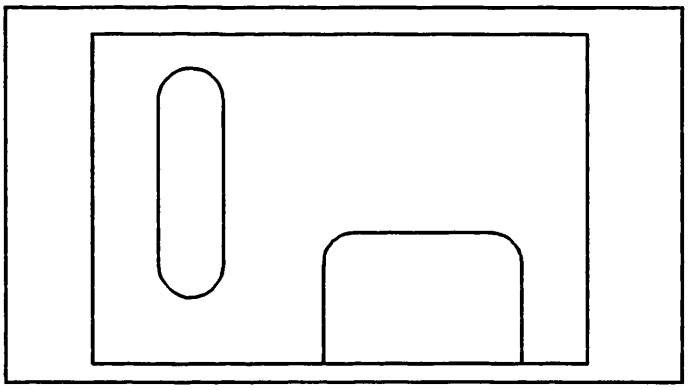


Figure 6.39 More complex pocket shapes

In this example, a closed pocket consisting of two 180 deg arcs interconnected with lines, as well as a side pocket consisting of two 90 deg arcs also interconnected with lines need to be machined. The previously mentioned algorithm clearly could not cope with this type of situation, or in general, situations involving arcs interconnected with lines. A new approach was therefore called for.

After a lot of experimentation, it was decided to focus on what happened to the start and end points of each newly discovered arc(s). In a basic situation, as in **Figure 6.38**, if these two points do not have any lines attached to them, and if the arc is 90 deg it forms an open pocket, whereas if it is 180 deg it will form a side pocket. If however, these points had lines starting from or ending at them, then the opposite ends of these lines should be examined. These might end at another arc (hence indicating a closed pocket as in **Figure 6.39**), or perhaps at an edge of the component. The need to accurately determine the start and end points of the identified arcs is thus evident and lead to the development of the algorithm previously described in section 6.5.2.

Once the start and end point determination algorithm was in place, a pocket "logic table" was built. This decides the type of pocket based on the number of arcs and lines that the pocket might consist of. In its current form the table is as follows:

No. of objects	Arcs	Lines	90 deg	180 deg
1	1	0	Open	Side
2	1	1	Open	Open
3 A	2	1	Side	-
3 B	1	2	Open	Side
4	2	2	-	Closed
5	2	3	Side	-
6	-	-	-	-
7	-	-	-	-
8	4	4	Closed	-

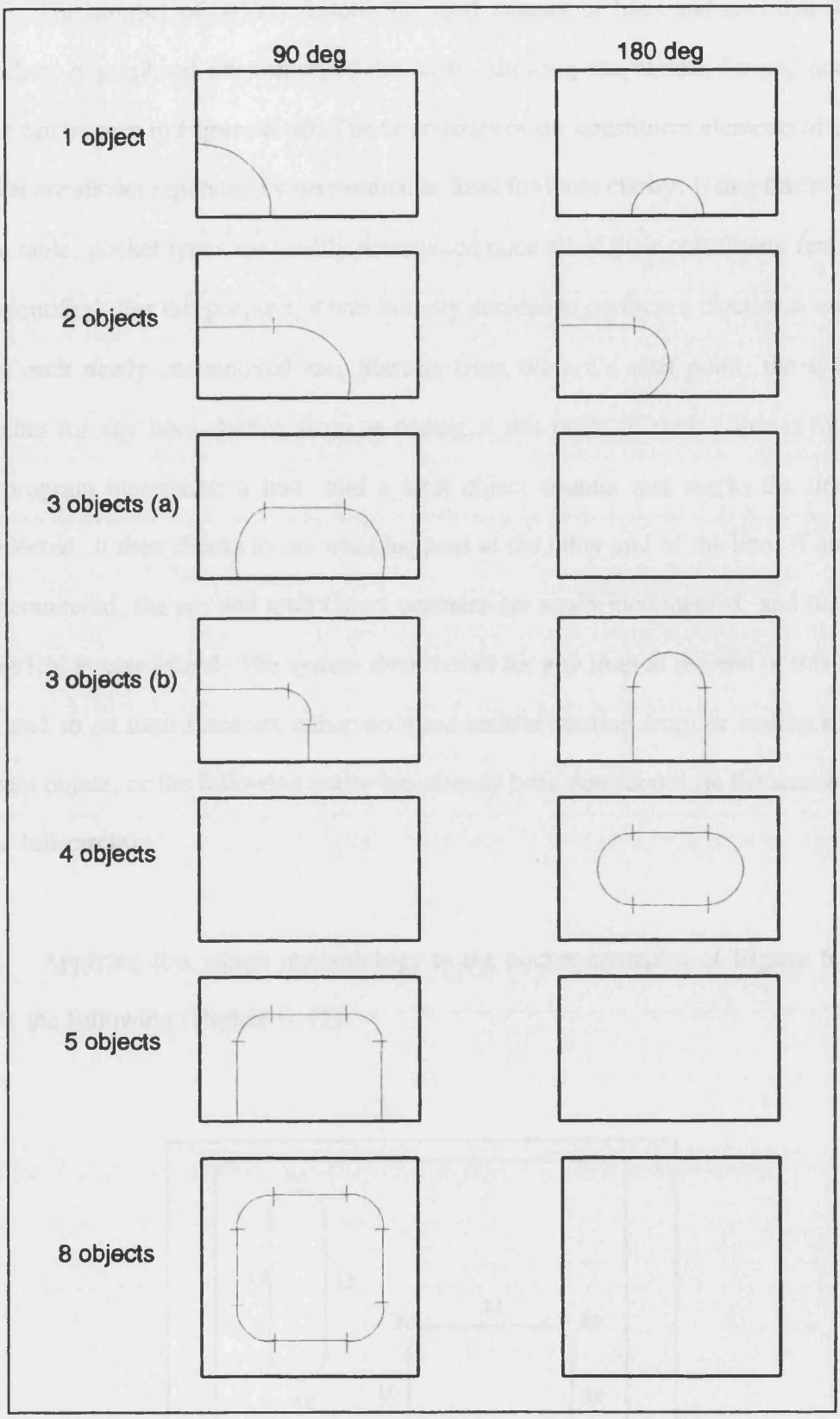


Figure 6.40 A graphical representation of the pocket logic table

The number of objects denotes the total amount of lines and arcs that form a pocket. A graphical illustration of this table, showing the various formed pocket types can be seen in **Figure 6.40**. The boundaries of the constituent elements of each pocket are shown separated by perpendicular lines for more clarity. Using this pocket logic table, pocket types are readily determined once all of their constituent features are identified. For this purpose, it was initially decided to perform a clockwise search from each newly encountered arc. Starting from the arc's start point, the system searches for any lines starting from or ending at this point. If such a line is found, the program increments a line and a total object counter and marks the line as considered. It then checks to see what happens at the other end of the line. If an arc is encountered, the arc and total object counters are again incremented, and the arc is marked as considered. The system then checks for any lines at the end of this new arc, and so on until there are either no more entities starting from or ending at the current object, or the following entity has already been considered (ie the search has gone full circle).

Applying this search methodology to the pocket examples of **Figure 6.39**, gives the following (**Figure 6.41**):

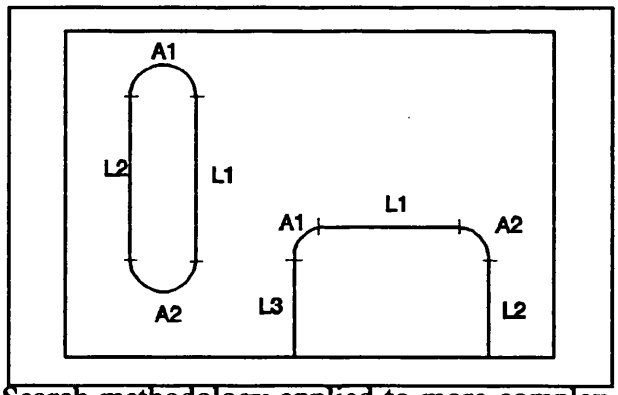


Figure 6.41 Search methodology applied to more complex pocket shapes

For the indicated closed pocket, if either arc A1 or A2 were first encountered, the clockwise search would yield (say for A1) line L1, arc A2 and line L2 before the considered arc A1 was again encountered. However, for the side pocket, if arc A2 was first encountered, line L2 would next be found and the search would stop, since there are no more entities following L2. An erroneous pocket reporting would thus be made.

To correct this situation, it was decided that the system should also perform an anti-clockwise search from the first encountered arc, looking for any entities not already considered. Therefore, if this search was performed on the closed pocket, nothing would be found since all the entities there would have been marked by the clockwise search. If however, this search was performed on arc A2 of the side pocket, line L1, arc A1 and line L3 would now be discovered, and the pocket type would be correctly deduced.

Following the results of both a clockwise and anti-clockwise search for a newly encountered arc, the program has the following information for the potential current pocket:

(a) the total number of objects that the pocket consists of (from object counter)

(b) the type and number of arcs that the pocket consists of (from arc counter)

(c) the number of lines that the pocket consists of (from line counter).

The pocket logic table is queried, and a correct answer for the current pocket type produced.

Once the algorithm for determining a pocket's type was developed, it was

then enhanced to resolve the pocket's particular characteristics: these include its status (normal or "hidden"), as well as its length, width and depth. These characteristics are indicated graphically in **Figure 6.42** below:

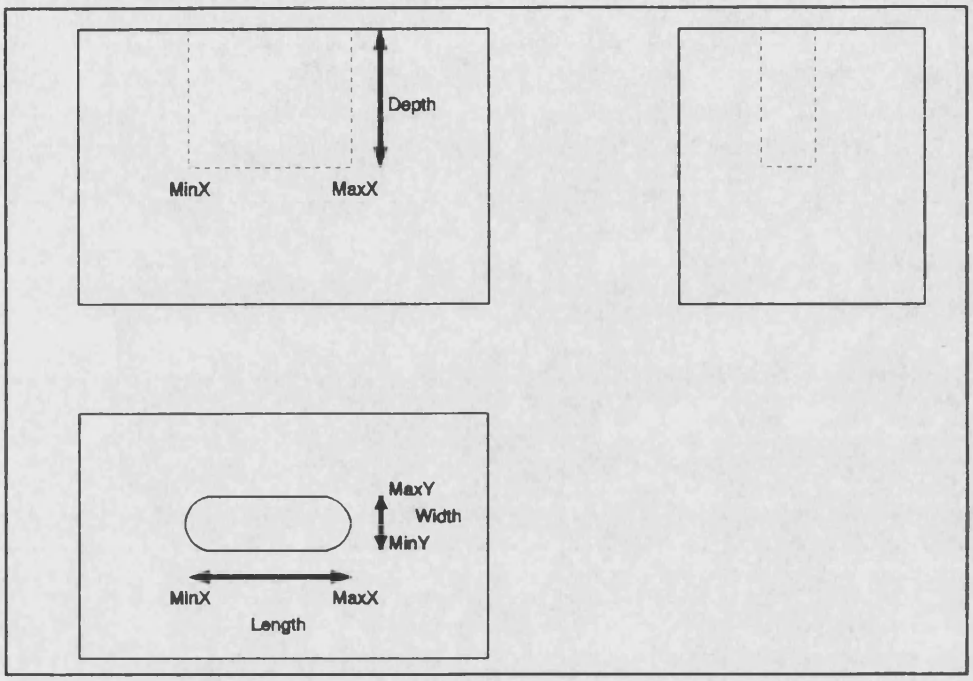


Figure 6.42 Convention used for dimensioning a pocket

The length of the pocket in this context can be defined as its dimension in the horizontal plane (including any radii of its constituent arcs), while the pocket's width can be defined as its dimension in the vertical plane (again including any radii). The pocket's depth is self-explanatory.

The issue of whether a pocket is normal or "hidden" is easily solved. As mentioned previously, if the discovered arc is "hidden", then it follows that the entire pocket is also hidden. This has repercussions on the pocket's depth, as explained later.

In order to solve the problem of determining the pocket's length, two variables, MinX and MaxX, were created. These keep track of the current pocket's minimum and maximum X points in a similar manner to the approach used for determining a layer's minima and maxima. In other words, with each newly discovered entity (line or arc) determined as belonging to the current pocket, MinX and MaxX are updated with the new entity's "extreme" values (its Xstart or Xend) if they are smaller/larger than the MinX/MaxX values determined so far. A special clause is used in the case of horizontal 180 deg arcs (as in **Figure 6.42** above), that is arcs that start or end along AUTOCAD's 90/270 line. In this case the radius R is subtracted from/added to the MinX/MaxX values to provide the correct "extreme" X values.

Once the search process for a pocket's entities is complete, MinX and MaxX contain the values of the pocket's horizontal minimum and maximum points. The length of the current pocket is then simply MaxX - MinX.

A similar approach is used for determining the current pocket's width. Two variables, MinY and MaxY were created and updated in the same way with each newly discovered entity as MinX and MaxX, but this time using the entity's Y values. The special clause is again used in the case of vertical 180 deg arcs, that is arcs that start or end along AUTOCAD's 0/180 line. In this case the radius R is subtracted from/added to the MinY/MaxY values to provide the correct "extreme" Y values. Again, when the search process is completed, MinY and MaxY contain the values of the pocket's vertical minimum and maximum points. The width of the current pocket is then simply MaxY-MinY.

Attempting to resolve the depth of a pocket presented a more difficult challenge. In contrast to the feature identification problems tackled so far, which examined each layer-view of a drawing separately, a solution for this problem required the correlation of different views. In other words, information from one layer would have to be applied to another one for successful results.

Following considerable experimentation, a solution to this situation was eventually developed. The convention adopted uses the plan view for determining the depth of front view pockets, and the front view for determining the depth of plan view or end view pockets. In the example shown in **Figure 6.42**, the front view would thus be used.

The system then goes through the sorted (but not "streamed") line entities of the front view. It looks for a line (DASHED or CONTINUOUS) that has its Xstart=MinX of the pocket and its Xend=MaxX of the pocket. This line forms essentially the "bottom" of the pocket. When such a line is found, the depth of the pocket can then be easily deduced by the formula

Depth = Ymax front layer - Ystart of "bottom" line.

If however, the pocket is "hidden", then it would be represented in a drawing as in **Figure 6.43** below.

In this case, the depth of the pocket would be

Depth = Ystart of "bottom" line - Ymin front layer.

It should be noted that using first angle projection rules, if a pocket is found at the front view, these formulae would be reversed (ie the depth of a normal pocket in the

front view would be Ystart of "bottom" line - Ymin plan layer). Also for the same reason, the depth of a pocket lying in the end view would have to be calculated in the X plane, ie for a normal pocket it would be Xstart of "bottom" line - Xmin front layer. The previously described methodology is put into practice as follows:

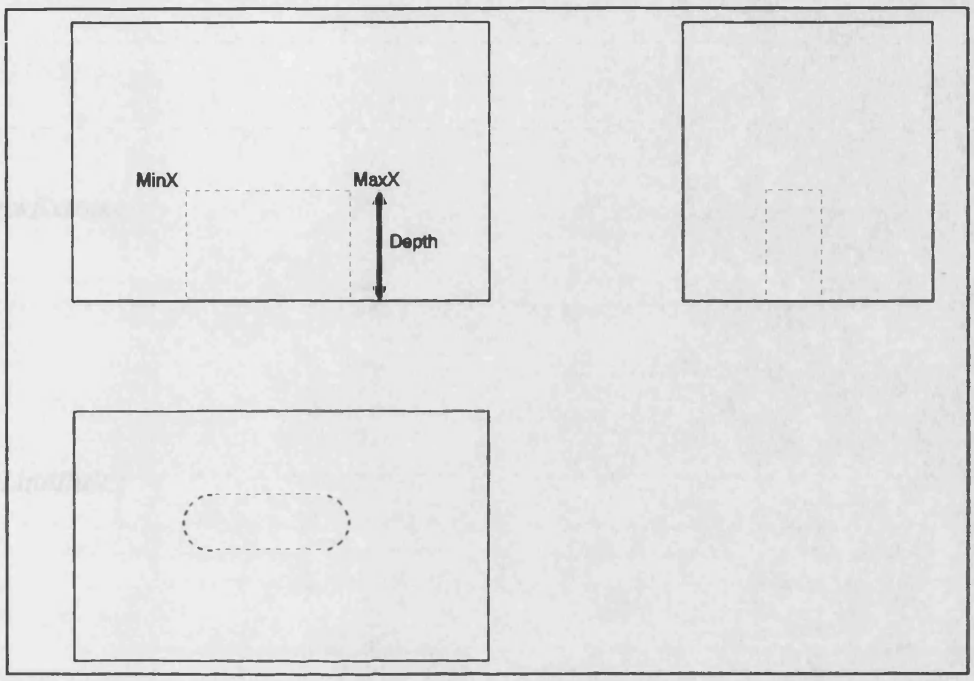


Figure 6.43 Calculating the depth of a hidden pocket

Each arc and its associated parameters is first read in from the interpreted data file using function *StoreArcProperties* from **props.c**. This function is responsible not only for reading and classifying arc entities and their characteristics, but also for determining their included angle (this is equal to endangle - startangle) as well as the coordinates of their start and end points using the algorithm described in section 6.5.2. Each arc, along with all its parameters, are next stored in an appropriate data structure for further processing.

The pocket identification task is performed solely by **pockets.c**. This consists of the following functions:

Pocket — The main part of the program. Determines the constituent objects of a pocket and hence the pocket type. Also calculates the pocket's dimensions and reports the discovered pocket(s) and all their parameters.

CheckExtremes — Calculates and updates (if necessary) the current pocket's MinX, MaxX, MinY, MaxY so that its dimensions can be determined.

GetLineIndex — Confirms that the current pocket contains a line. Identifies its direction and marks it as considered. Increments the line and total object counters for this pocket.

GetArcIndex — Confirms that the current pocket contains another arc. Marks it as considered. Increments the arc and total object counters for this pocket.

Program **pockets.c** starts by implementing its main function *Pocket*. This starts by looking for 90 or 180 deg arcs (implying the existence of pockets) on the front layer. If such an arc is found, then it is marked as considered, and the arc and total object counters are set to 1. Function *CheckExtremes* for this arc is next executed. This calculates the current pocket's "extreme" values (MinX, MaxX, MinY, MaxY) so that its dimensions can be determined.

The system now performs a clockwise direction search. It checks to see if the arc has a line entity starting from (or ending at) its start point using function *GetLineIndex*. If such a line is found, it is marked as considered, and the line and total object counters are incremented. Function *CheckExtremes* is again implemented for updating the pocket's "extreme" values, if required. The system next proceeds to check if another arc exists at the line's start/end points using function *GetArcIndex*. If such an arc is discovered, it is marked as considered, and the arc and total object counters are incremented. Function *CheckExtremes* is again employed for updating, if necessary, the pocket's "extreme" values.

This procedure is repeated until there are no more line or arc entities belonging to the current pocket to be found. The program then reverts back to the original arc that it started from, and now performs an anti-clockwise direction search using the same methodology. When this is completed, the program has at its disposal:

(a) the total number of objects that the pocket consists of (from object counter)

(b) the type and number of arcs that the pocket consists of (from arc counter)

(c) the number of lines that the pocket consists of (from line counter).

(d) the current pocket's status (normal or hidden).

The pocket logic table is now queried in order to decide the type of the pocket under consideration. Once this has been decided, the algorithm for determining the pocket's depth is next executed. In this case, it looks for a "bottom" line amongst the plan view entities, and calculates the pocket's depth according to the stated rules. The identified pocket can now be reported. The pocket reporting format is as follows:

"(Status)(type) Pocket on Layer (layernumber).

Width=(width), Length=(length), Radius=(radius), Depth=(depth).

Arc (arcnumber): X Centre=(Xcentre), Y Centre=(Ycentre)"

where

(Status)= the pocket's status. This entry remains blank if the pocket is normal, or displays "Hidden" if the pocket is hidden.

(type)= the pocket's type (open, side or closed).

(layernumber)= the layer number that the pocket belongs to.

(width)= the pocket's width (MaxY-MinY) (in mm).

(length)= the pocket's length (MaxX-MinX) (in mm).

(radius)= the radius of curvature of the pocket. This is equal to the radius of curvature of the detected arc(s).

(depth)= the pocket's depth (in mm).

(arcnumber)= the constituent arc(s) that form the pocket and their centres' coordinates are presented here sequentially. This is useful for accurately locating the pocket on the component's surface.

(Xcentre)= the X coordinate of the centre of a constituent arc of the pocket.

(Ycentre)= the Y coordinate of the centre of a constituent arc of the pocket.

Once the pocket is properly reported and recorded in the output file, the program proceeds with the next unconsidered 90/180 deg arc for identifying the next pocket (if applicable). A flowchart of the general operation sequence of the **pockets.c** program can be seen in **Figure 6.44**. When all pockets in the front layer are identified and reported, the program continues the same process with the plan and then the end layers. Once all pockets in all layers are completely extracted, the program ends execution.

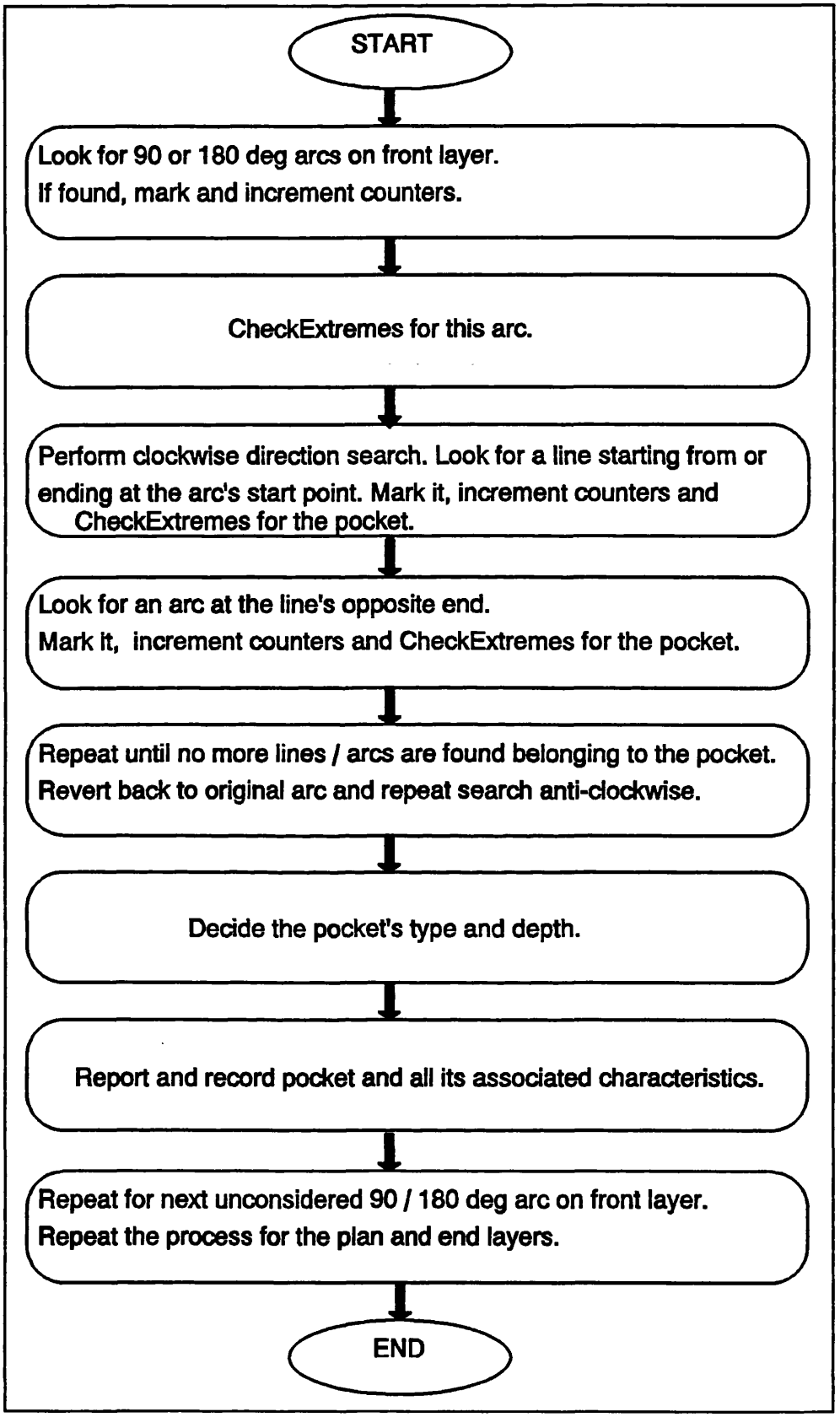


Figure 6.44 Operation sequence flowchart for **pockets.c**

CHAPTER 7

IDENTIFYING CYLINDRICAL MANUFACTURING FEATURES AND DIMENSIONAL TOLERANCES

7.1 Identifying Cylindrical Features

7.1.1 Introduction

Cylindrical manufacturing features, according to the convention used by [Rustom 1992], include various types of holes as well as threads. As it can be seen in **Figures 6.1** and **6.2** from **Chapter 6**, the holes are classified as plain (primary cylindrical feature) or stepped and countersunk (secondary cylindrical features). BUFIP adopts the same convention, with the difference that countersunk holes are also classified as stepped holes. Hence, based on the definition used in BUFIP, stepped holes can be either countersunk (conical stepped) or counterbored (plain stepped).

In addition, BUFIP further discriminates each type of hole into three sub-categories according to their bottom end:

(i) Holes that go completely through the component (termed through holes).

(ii) Holes that have a flat bottom end (in practice these are more difficult to produce and less common).

(iii) Holes that have a conical shaped end (conical ended holes, the most common in practice. They are easier to drill, since the drill itself is conical ended).

Figure 7.1 below shows an engineering drawing of a component containing three plain holes on the plan view, each with a different bottom end, as seen in the front and end views.

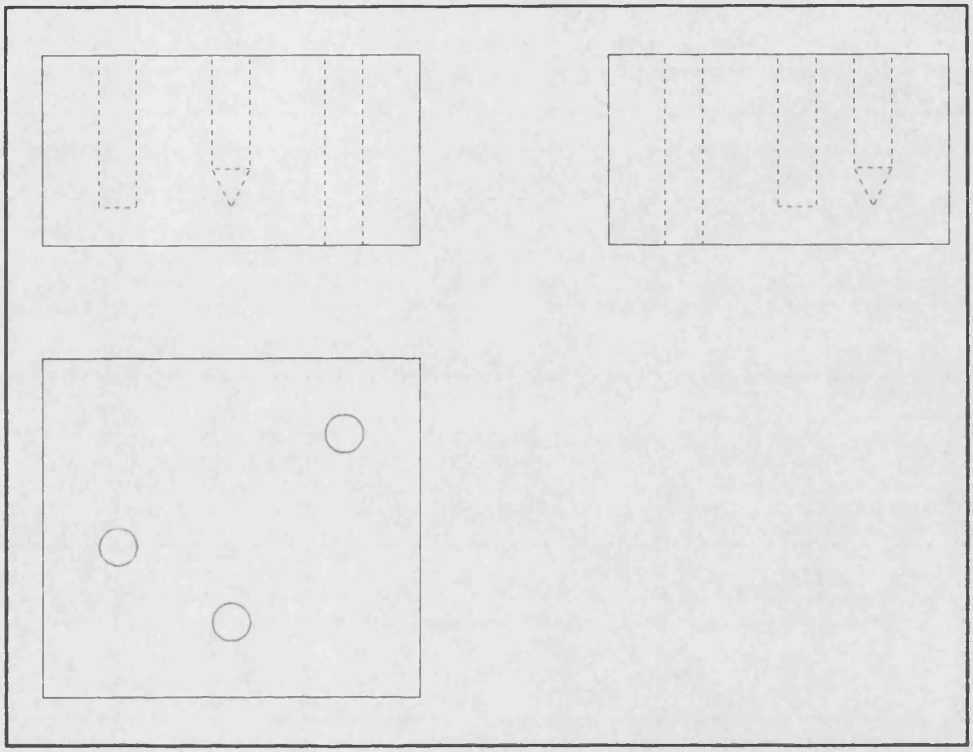


Figure 7.1 The three types of plain holes

Stepped holes in engineering drawings are represented by two concentric circles if viewed from the top. Their profiles vary depending on whether they are countersunk or counterbored. **Figures 7.2** and **7.3** show engineering drawings of stepped countersunk and counterbored holes with different bottom ends. Obviously such holes have two different diameters and two different depths.

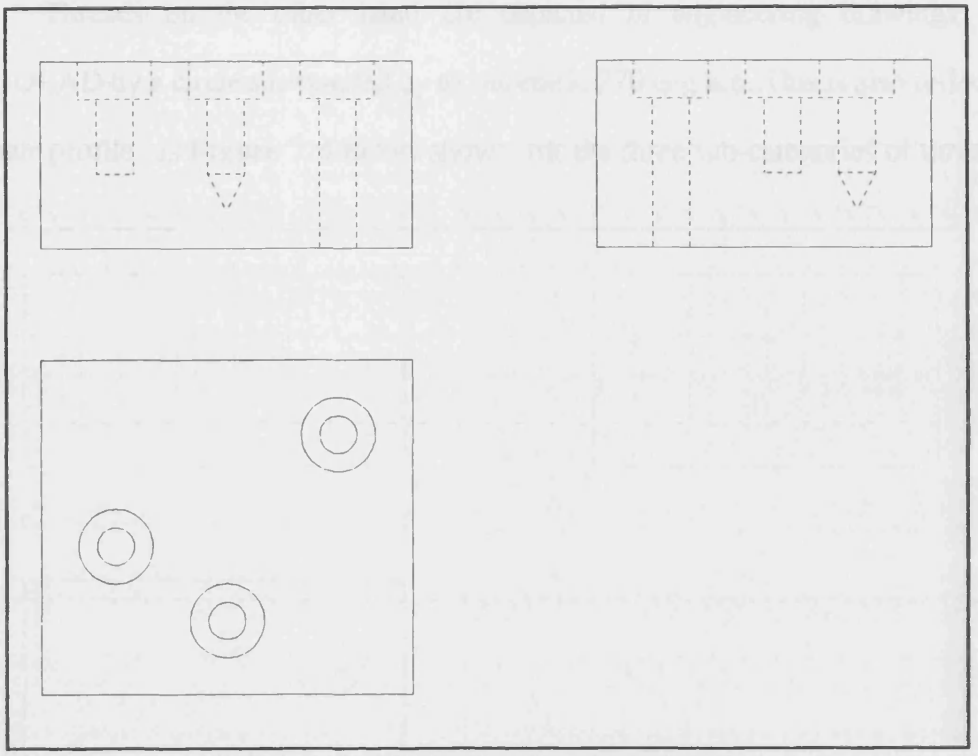


Figure 7.2 Stepped counterbored holes

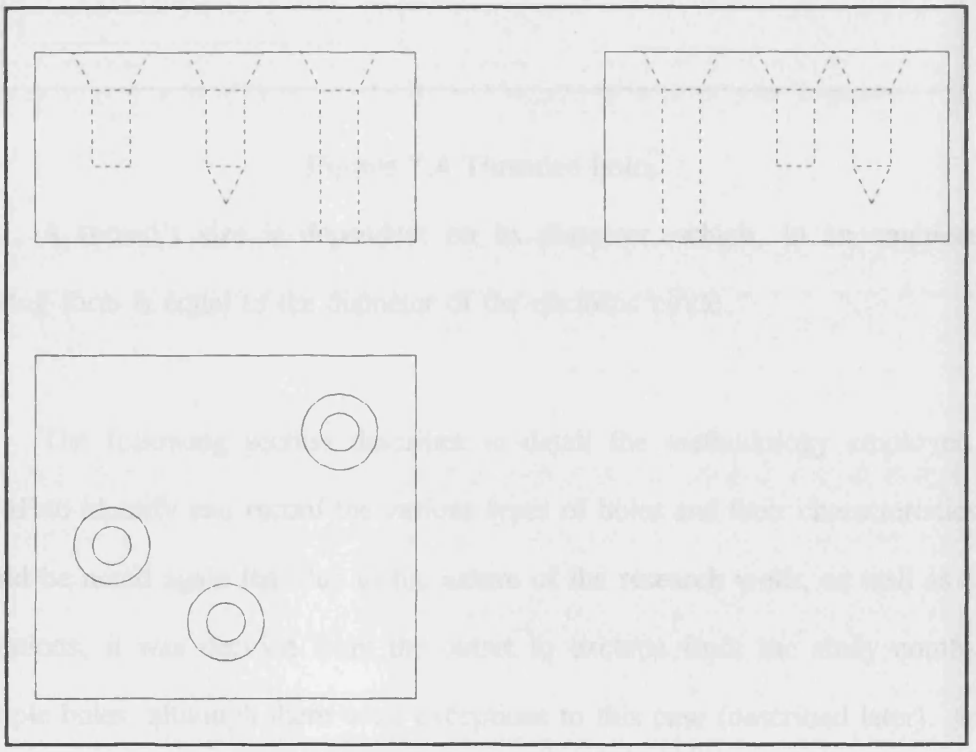


Figure 7.3 Stepped countersunk holes

Threads on the other hand are depicted in engineering drawings and AUTOCAD by a circle surrounded by a concentric 270 deg arc. This is also reflected in their profile, as **Figure 7.4** below shows, for the three sub-categories of threads:

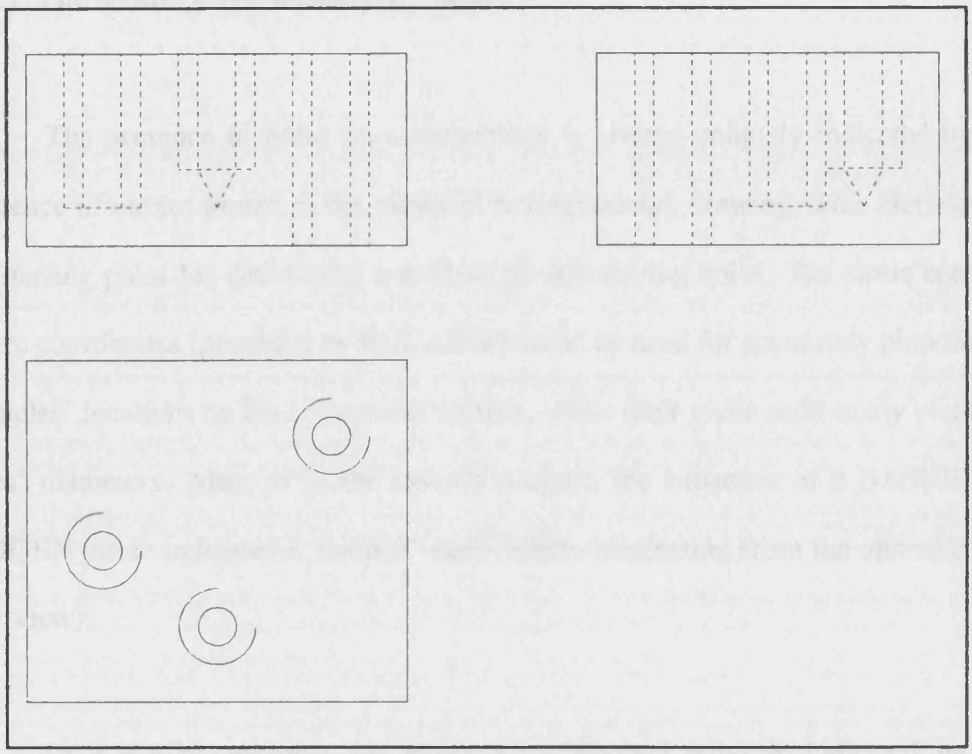


Figure 7.4 Threaded holes

A thread's size is dependent on its diameter, which, in an engineering drawing form is equal to the diameter of the enclosed circle.

The following section describes in detail the methodology employed by BUFIP to identify and record the various types of holes and their characteristics. It should be noted again that due to the nature of the research work, as well as time limitations, it was decided from the outset to exclude from the study combined multiple holes, although there were exceptions to this case (described later). Also, hole profiles were not allowed to completely and exactly overlap, but the system

should be able to cope with various other degrees of overlap.

7.1.2 The strategy for identifying holes

The presence of holes on a component is always uniquely indicated by the existence of circles in one of the views of its engineering drawing. This fact formed the starting point for developing a method for identifying holes. The circle entities' centre coordinates (provided by BUCADIP) could be used for accurately pinpointing the holes' locations on the component surface, while their given radii easily yield the holes' diameters. Also, as in the case of pockets, the existence of a DASHED or HIDDEN circle indicated a "hidden" hole (a hole originating from the opposite side of a view).

In a similar manner, the presence of stepped holes is indicated by the existence of two concentric circles. The inner circle's radius provides the diameter of the smaller hole, while the outer circle's radius gives the diameter of the step. Similarly, threads are denoted by the presence of a circle and a concentric 270 deg arc. This is convenient for distinguishing between arcs forming threads and arcs forming pockets, since the latter are formed by 90 or 180 deg arcs only, as described in **Chapter 6**. The thread's diameter is again derived from the included circle's radius.

Once the methodology for identifying the existence of holes, their type and their diameters was in place, it was time to examine how their depths as well as the

type of their bottom ends could be calculated. For this purpose it was obvious that data from different views of the drawing would have to be correlated (as in the case of pockets). It was decided to adopt the same convention for correlating views with the one used for pockets: that is, the plan view being used for front view holes, and the front view being used for plan and end view holes.

Consider a plain, flat-bottomed hole in the top of a component, that is depicted on the front layer of a drawing as in **Figure 7.5**; as can be seen from **Figure 7.5**, the hole is represented in the front view by a dashed line profile (the profiles of holes will always be dashed in contrast to pockets which might be dashed or continuous) consisting of the lines L1, L2, L3. These are already sorted in a top-to-bottom, left-to-right order, as explained in **Chapter 5**.

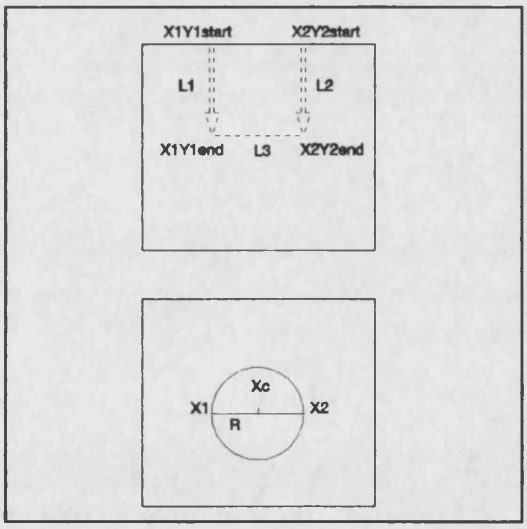


Figure 7.5 Plain flat-bottomed hole

Now, if the radius of the circle is subtracted from the given Xc coordinate of its centre, coordinate X1 would be obtained. This is the same X coordinate that line

L1 has to start from, and end to as well. The program could therefore be made to look for a vertical dashed line (same X coordinates) starting at X1. Once such a line is found (L1 in this case), then the depth of the hole would simply be

Depth = Y1start - Y1end.

The correct identification of the hole could be further verified by finding line L2. This is offset in the X plane from line L1 by the diameter of the hole (2R).

The above algorithm also applies to "hidden" holes. If the hole in **Figure 7.5** was hidden, then it would be represented as in **Figure 7.6** below:

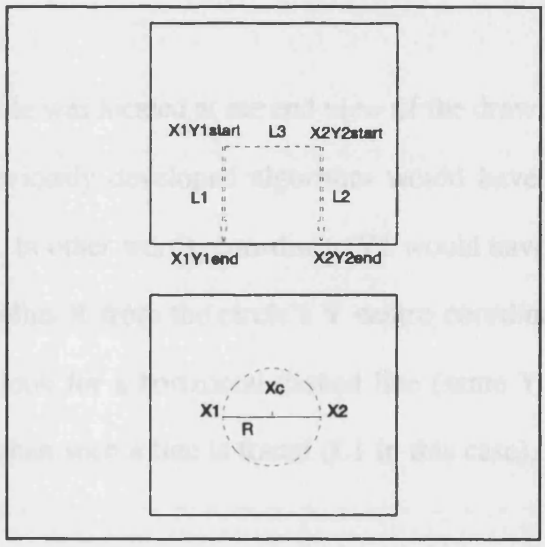


Figure 7.6 Hidden plain flat-bottomed hole

As it can be seen from **Figure 7.6** its depth would still be calculated as

Depth = Y1start - Y1end

since the lines are always sorted in a top-to-bottom order.

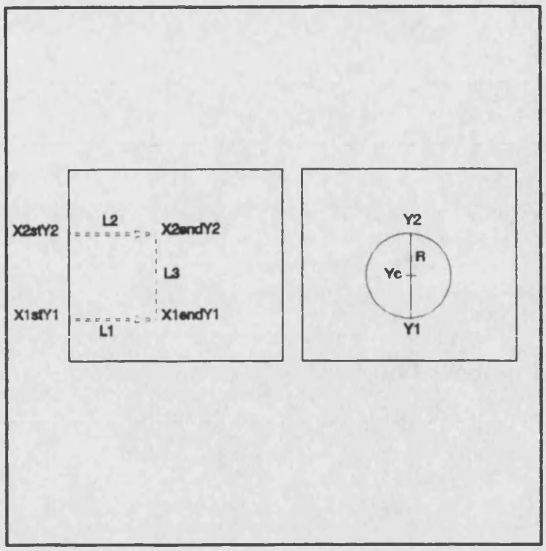


Figure 7.7 End layer plain flat-bottomed hole

If the same hole was located at the end view of the drawing (**Figure 7.7**) then in this case, the previously developed algorithm would have to be applied in the horizontal (X) plane. In other words, coordinate Y1 would have to be calculated first by subtracting the radius R from the circle's Y centre coordinate Yc. The program would then have to look for a horizontal dashed line (same Y coordinates) starting and ending at Y1. When such a line is found (L1 in this case), the depth of the hole would now be

Depth = X1end - X1start.

As previously, this algorithm also applies to "hidden" holes. Therefore, the system would have to look for the profile lines in the vertical Y plane for holes belonging to the front or plan view layers, and in the horizontal X plane for holes belonging to the end layer.

Additionally, if the calculated depth was equal to Ymax - Ymin (or Xmax - Xmin for end view holes) for the layer, this indicates that the hole passes through the component (a through hole). On the other hand, if the hole was conically ended, its depth would still be derived from the above formulae, since the depth of a conical ended hole is usually measured down to its flat bottom, and not the tip of the cone. A method thus had to be found though, to determine if a hole was conical ended or flat.

A solution to this problem eventually emerged, based on a similar search pattern to the one used for determining a hole's depth. Consider the case of a conically ended plain hole located on the plan layer (**Figure 7.8**).

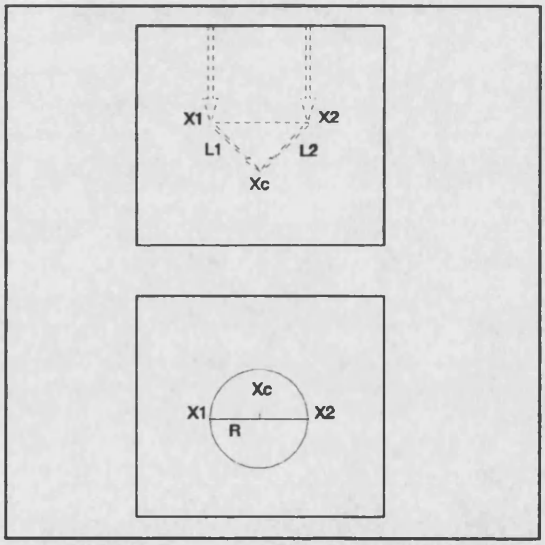


Figure 7.8 Plain conical ended hole

By identifying points X1 and X2 as mentioned previously, the system would now have to search for a dashed line starting from X1 and ending at Xc (the X coordinate of the centre of the circle). If such a line is found (L1 in this case), then the hole has a pointed tip (conical ended), otherwise it is flat-bottomed. The conical

end can be verified by looking for another dashed line, starting from Xc and ending at X2 (line L2). As previously, the search would have to be performed in the horizontal plane for holes located on the end layer of the drawing. Also, hidden holes would again use the same algorithm. Once the algorithms for determining the depth and bottom end type of plain holes were in place, it was time to expand them to include more complicated hole types.

Starting with threads, the same algorithms could be used without any modifications, since thread measurements, as mentioned previously, are based on their included circle, which for all intents and purposes behaves exactly as a plain hole.

Stepped holes were resolved by introducing a more expanded version of the previous search algorithms. Consider the case of a stepped counterbored hole, located on the plan view of a drawing (**Figure 7.9**):

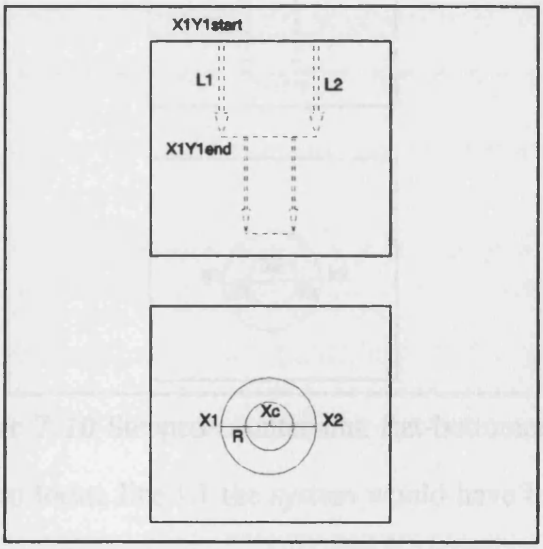


Figure 7.9 Stepped counterbored flat-bottomed hole

By deducting the radius R of the outer circle (instead of the radius of the inner circle) from the centre coordinate Xc, point X1 can be determined. The system could then switch to the front view and search for a dashed line starting from X1Y1start and ending at X1Y1end. This corresponds to the depth profile line of the step. Line L1 would thus be detected, and the depth of the step could be calculated in a similar way to the depth of a plain hole, ie

Depth of counterbore step = Y1start - Y1end.

The depth of the inner hole as well as its bottom type can be found by the same procedure used for plain holes.

Countersunk holes require a slightly altered version of the algorithms. Consider a stepped countersunk hole located on the plan view of a drawing (**Figure 7.10**).

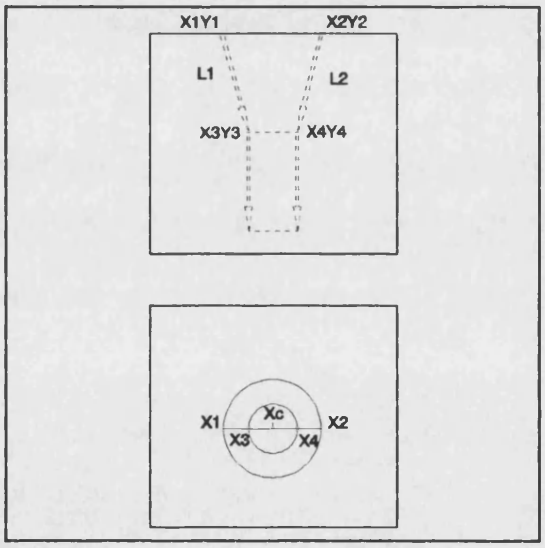


Figure 7.10 Stepped countersunk flat-bottomed hole

In this case, to locate line L1 the system would have to search for a dashed line starting from X1Y1 (outer circle) and ending at X3Y3 (inner circle). The depth

of the step would then be

Depth of countersunk step = Y1 - Y3.

The depth of the inner hole as well as its bottom type can be found by the same procedure used for plain holes.

Therefore, to establish the depth and type of a stepped hole, the system first searches for a dashed line indicating a counterbore step. If no such line is found, the system then searches for a dashed line denoting a countersunk step, and its depth would be calculated accordingly. As previously, the search for stepped hole profiles would have to be performed in the horizontal plane if the stepped hole is located on the end layer. Also, these algorithms apply to "hidden" stepped holes as well.

Following the application and testing of the above algorithms in code, the issue of a hole type combination was examined. It is common engineering practice to have stepped holes (usually counterbored) with threaded inner parts. In this way, the screws or bolts that fit into the threads can sit "flush" with the component's surface. **Figure 7.11** shows such a hole on the front layer of a component. It was therefore felt necessary, despite the exclusion of combined hole types from the scope of the research, to have the system identify and report such threaded stepped holes.

The software was thus updated to recognise the existence of threads within stepped holes. This involved looking for a concentric 270 deg arc lying within two concentric circles, as **Figure 7.11** shows. The arc's radius of curvature should be greater than the radius of the inner circle, but less than the radius of the outer circle. The algorithms for calculating the thread and the step depths, as well as the bottom

end type remain unaffected.

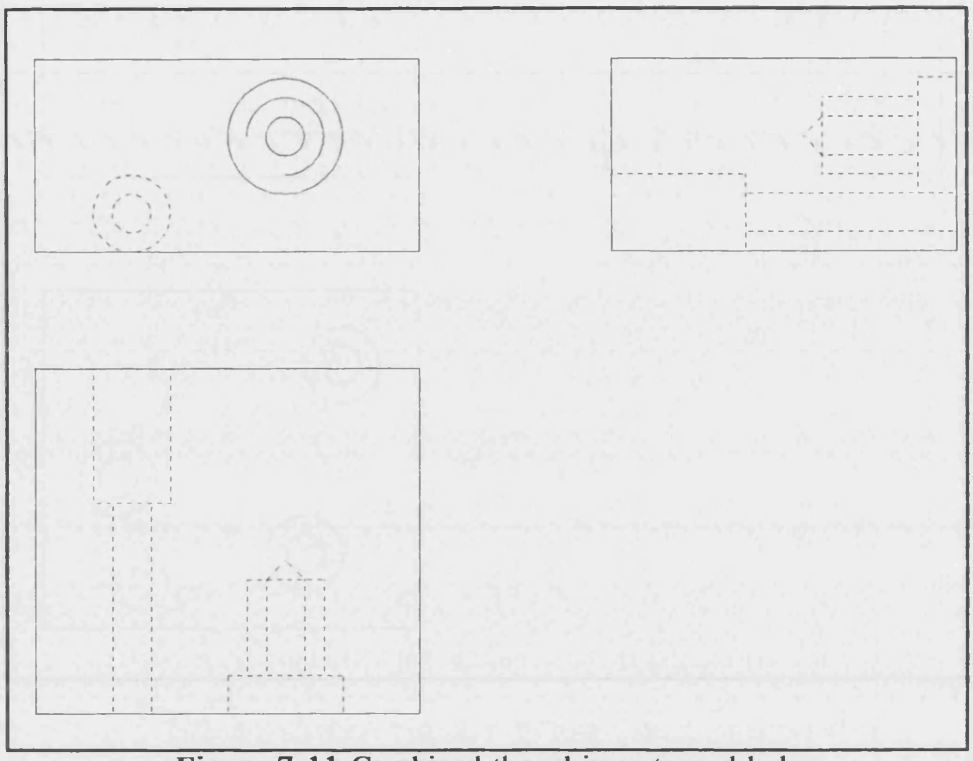
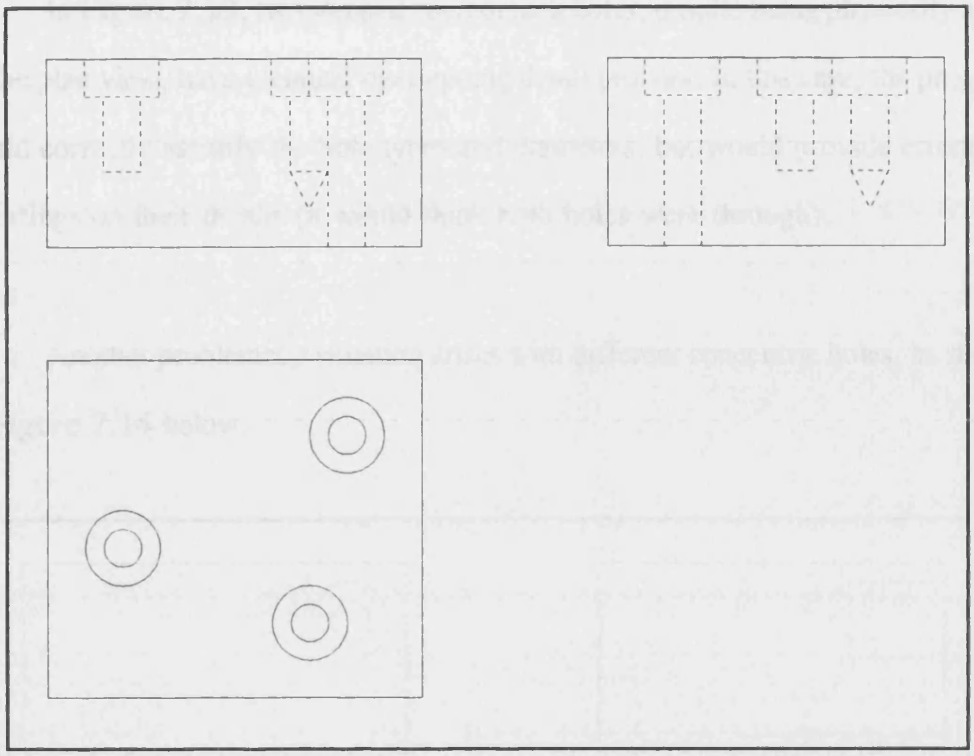


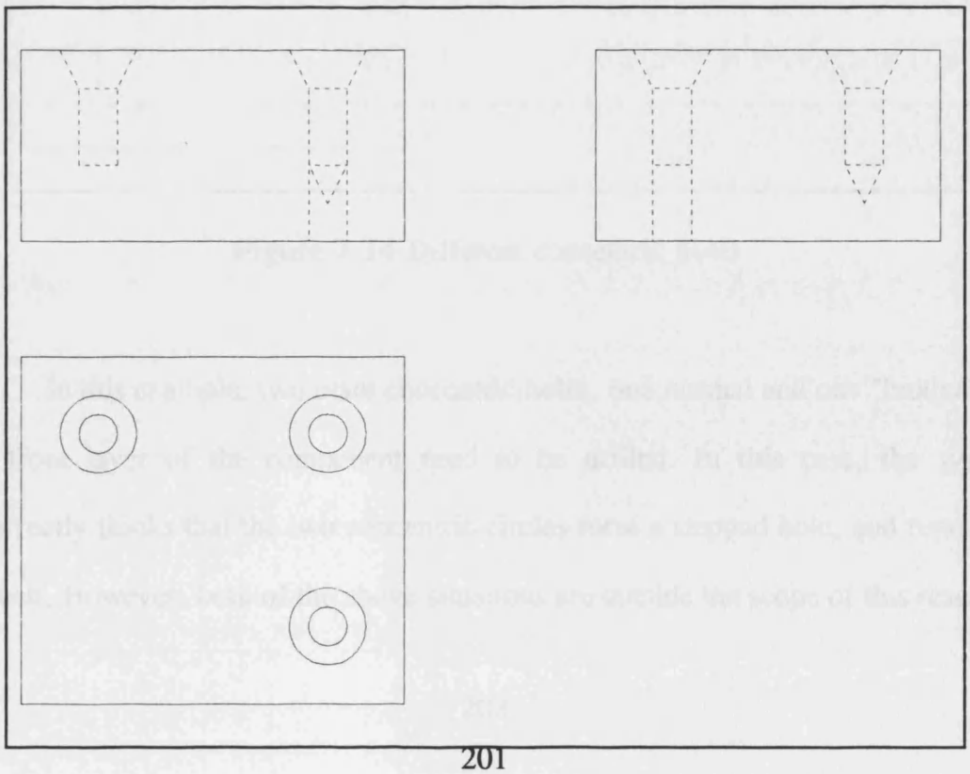
Figure 7.11 Combined thread in a stepped hole

Another issue experimented upon was hole profile overlap. Whenever two or more holes exist on a component's surface, it is sometimes unavoidable that their depth profiles overlap. Consider for example **Figure 7.12** below where two stepped counterbored holes on the plan view of the component need to be machined.

Despite their separate physical locations on the plan view, their depth profiles overlap. This does not pose any problems for the system however, since it looks for depth profile lines at accurately predetermined locations. In fact, it was discovered after extensive experimentation that the system and the developed algorithms could cope admirably with any type of partial depth profile overlap at any layer of the drawing. It only reached its limits when two different holes had depth profiles exactly overlapped, as in **Figure 7.13** below.



Figures 7.12 and 7.13 Hole depth profile partial and exact overlap



In **Figure 7.13**, two stepped countersunk holes, despite being physically apart on the plan view, have identical overlapping depth profiles. In this case, the program would correctly identify the hole types and diameters, but would provide erroneous reportings on their depths (it would think both holes were through).

Another problematic situation arises with different concentric holes, as shown in **Figure 7.14** below.

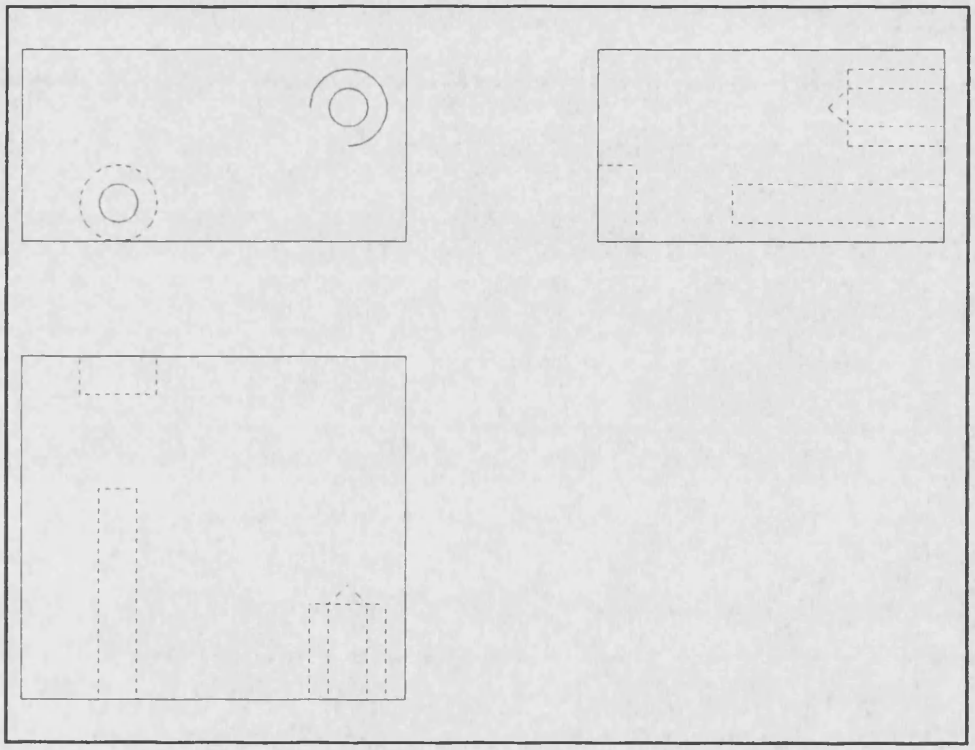


Figure 7.14 Different concentric holes

In this example, two plain concentric holes, one normal and one "hidden" on the front layer of the component need to be drilled. In this case, the system incorrectly thinks that the two concentric circles form a stepped hole, and reports it as such. However, both of the above situations are outside the scope of this research

(as mentioned in the introduction), but are presented here to illustrate the current software's limitations on hole identification.

These current limitations could possibly be overcome if the user carefully selects which sides of the component will form the front, plan and end views of its drawing, in such a way that exact overlap is avoided in the front or plan views. Since the system searches for depth profiles in these two views only, it might be possible to draw a problematic component from such an angle so as to result in uncluttered front or plan view profiles (it does not matter what happens to the end view), and hence in correct hole extraction.

The above algorithms are applied in programming code as follows. Hole identification and classification is performed by **holes.c**. This consists of the following functions:

Hole: The main part of the program. Responsible for identifying and reporting all holes and their characteristics.

GetLinePair: Multi-purpose function. Identifies and records the pair of dashed lines forming the depth profile of a hole at the specified input coordinates. Also determines the existence of counterbore or countersunk holes with appropriate manipulation of its input parameters to look for the characteristic dashed lines forming these two hole types. Furthermore, it can also identify the existence of conical ended holes by looking for the characteristic dashed lines forming the cone.

GetLineIndex: Verifies that a dashed line exists at the specified input coordinates.

GetObjectIndexes: Identifies the type(s) of holes or threads existing at any layer using their characterising circle or 270 deg arc entities, and records their centre coordinates and their radii.

The main function *Hole* starts by looking for any circle entities in the interpreted file. If no such entities are found, no holes exist and the program stops. Provided there are circle entities, function *Hole* proceeds to define the layers for searching for depth profile lines, as well as the plane of search. Thus, for front layer holes, the plan view should be examined for depth lines; otherwise the front view should be checked. Also, for end layer holes, a horizontal X plane search should be performed, otherwise a vertical Y plane search should be made.

The next function *GetObjectIndexes* is then activated. This identifies the number and type(s) of holes or threads existing on any layer, by extracting their characterising circle and 270 deg arc information. The circle centre coordinates and radii values as well as threaded and hidden status are also recorded for further processing.

The program then continues with the first detected hole, and performs a search in the Y plane (if applicable). Using the described algorithms, it calculates the X and Y coordinates that the first dashed depth line should have. Function *GetLinePair* is then executed.

Function *GetLinePair* is a multi-purpose function. It was observed that there was a large similarity in the methodology of looking for different dashed lines, and that a single function could perhaps be employed for this task. Thus function *GetLinePair* was developed. By utilising appropriately calculated input coordinates, function *GetLinePair* can identify the pair of dashed lines forming the "sidewalls" of a hole, or step (lines L1 and L2 from the previous figures). For this purpose it employs the services of function *GetLineIndex*, which verifies that a dashed line indeed exists at the specified input coordinates. Function *GetLinePair* then calculates the coordinates of the second dashed line of the pair, and employs *GetLineIndex* again to verify its existence. Each hole depth profile is thus double-checked for correct identification.

By supplying appropriate input coordinates and offsets to *GetLinePair*, it can also be made to look for the pair of dashed lines forming a counterbore or countersunk step, as well as for determining whether a hole has a conical end. Another advantage is that the same function can also be employed for searching in the horizontal X plane, with modified input coordinates according to the developed algorithms. When *GetLinePair* completes its search, the results of its success or not, and (if applicable) the requested pair of discovered lines and their coordinates are recorded for further processing.

Returning with the results back to the main program, the depth of the hole can then be calculated according to the developed formula. If the hole is stepped, the program then proceeds to identify the step's characteristics. New input coordinates for the outer circle are generated, and the program again employs *GetLinePair* for

a counterbore situation. If the results of the search of *GetLinePair* are successful, then a counterbore stepped hole is declared, and its depth appropriately computed. Otherwise new values to the input coordinates are assigned, and *GetLinePair* tries for a countersunk step. Again, if the results are successful, a countersunk step is declared, and its depth appropriately computed.

The main program next checks for the bottom end type of the hole. If its depth, as mentioned previously, is equal to the component's side, then the hole is through. Otherwise, new input coordinates are assigned to *GetLinePair*, which now looks for the pair of dashed lines forming a hole's conical end. If the results of *GetLinePair* are successful, the hole bottom is declared as conically ended, otherwise it is declared as flat-bottomed.

The above process is now repeated for the horizontal X plane (if it is applicable instead), with appropriate changes in the input coordinates for *GetLinePair*, as well as in the depth calculation algorithms.

When the processing of the characteristics of the discovered hole is completed, the program then reports and records its findings in the output file. The report has the following format for each discovered hole:

"(Hidden)(Threaded)(Stepped)(Bottomtype) Hole found on layer (layernumber)
X Centre is (Xc), Y Centre is (Yc), Diameter is (dia), Depth is (d)
Step type is (steptype)
Diameter is (stepdia), Depth is (stepd) "

where:

(Hidden): Entry appears if hole is hidden, otherwise it remains blank.

(Threaded): Entry appears if hole is threaded, otherwise it remains blank.

(Stepped): Entry appears if hole is stepped, otherwise it remains blank.

(Bottomtype): The hole's bottom end. This can be either "Through", "Flat-bottomed" or "Conical Ended".

(layernumber): the layer number that the hole is located on.

(Xc): The hole's X centre coordinate.

(Yc): The hole's Y centre coordinate.

(dia): The hole's diameter (in mm).

(d): The hole's depth (in mm).

"Step type is... ": This entry appears only if the hole is stepped.

(steptype): The step's type. This can be either "Countersunk" or "Counterbore".

(stepdia): The step's diameter (in mm).

(stepd): The step's depth (in mm).

It should be noted that error reporting has been included in the code to trigger error messages when the program is unable to extract the characteristics of a discovered hole.

When the first detected hole is properly identified, and all its parameters extracted and recorded, function *Hole* proceeds with the next one, and so on until all holes in all layers have been processed. The program then ends execution. A flowchart of the general operation sequence of the program can be seen in **Figure 7.15.**

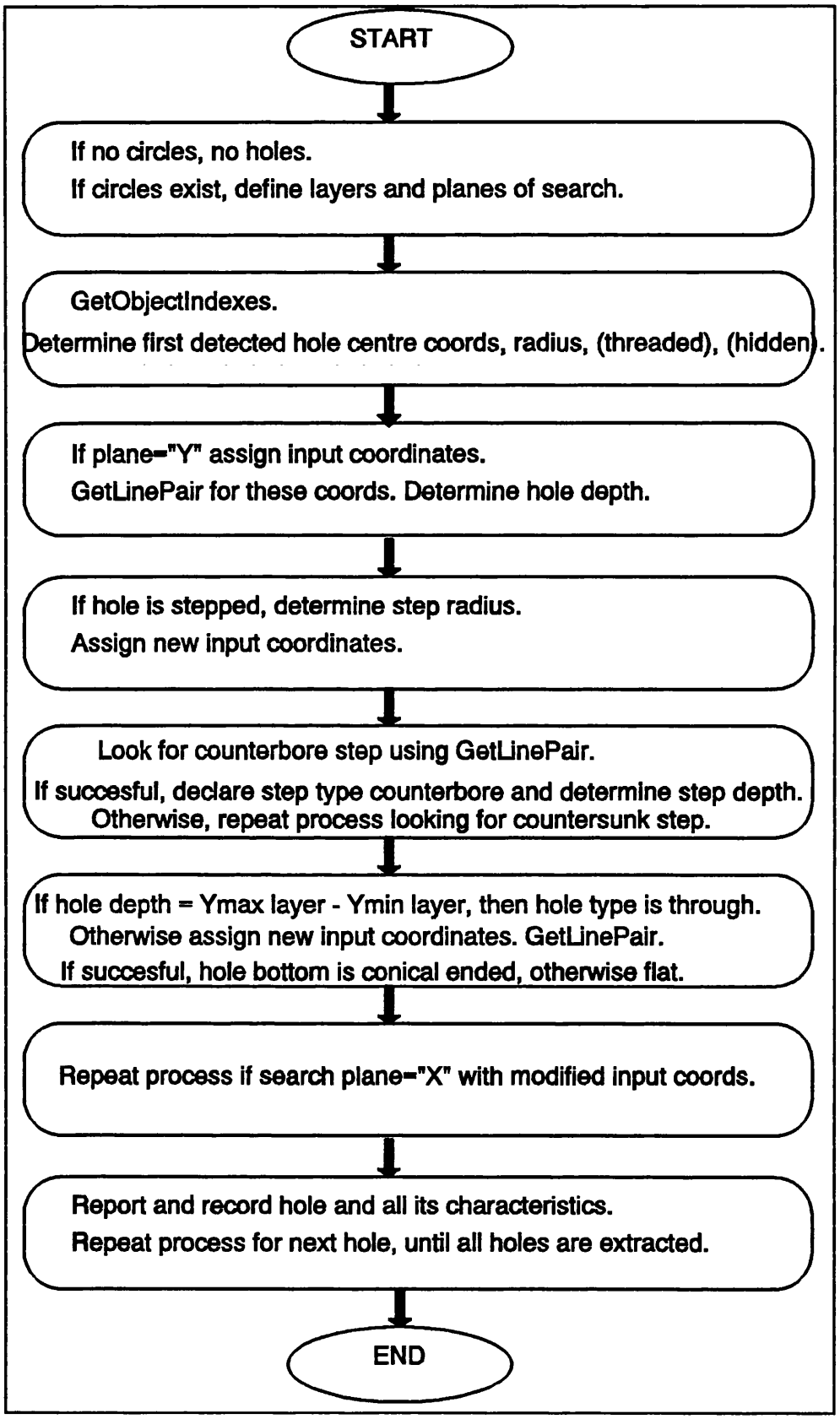


Figure 7.15 General operation sequence flowchart for **holes.c**

7.2 Identifying Dimensional Tolerances

7.2.1 Introduction

In many cases, component features are toleranced to provide greater precision. Tolerances show the limits within which a component's dimensions must be produced, and can take two main forms: dimensional and geometric. Dimensional tolerances, as the name suggests, deal with the limits placed on a component's desired nominal dimensions. Geometric tolerances on the other hand, specify the limits of geometric characteristics for the component's centrelines and surfaces. These include issues such as straightness, flatness, parallelism, concentricity etc.

Additionally, engineering drawings of components often include the desired quality of the surface texture, or surface roughness, that a component should have. This results from the method of machining adopted, as well as from machine vibrations or chatter. A series of preferred values of Ra in micrometers or an N series of numbers is used for the purpose of defining the limits of the qualities of surface roughness, along with a basic symbol for indicating the surface on which these values should be applied.

The following sections describe in detail the methodology adopted by BUFIP for identifying and extracting the dimensional tolerances of a prismatic part's overall dimensions. The developed solution, despite currently dealing with overall dimensions only, was designed to be expandable to cover a component's individual feature tolerances.

Once the dimensional tolerances are extracted, the component's minimum block shape envelope (the minimum block of raw material required to machine the component) can be accurately determined. The process for doing this is also discussed in section 7.2.2.

Geometric tolerance identification and surface roughness allocation has been excluded from the scope of this research work (as mentioned in **Chapter 1**) due to time limitations. However, some thoughts have been made on a possible strategy for their extraction and these are discussed in section 7.2.3.

7.2.2 The methodology for identifying dimensional tolerances

Dimensions and dimensional tolerances are depicted in engineering drawings using a set of established conventions and rules. **Figure 7.16** below shows a typical overall dimensioned prismatic component featuring a through slot.

As it can be seen from **Figure 7.16**, the representation of a dimension and its associated tolerance involves the drawing of several extra lines apart from the dimension and tolerance values themselves. These continuous lines (divided into "extension" and "dimension" lines separated by arrows) could unnecessarily overcomplicate the feature recognition process if left on the same layers as the other component entities. One of the first priorities for tolerance allocation therefore, was to separate the dimensioning from the normal entities of a component, by establishing an additional frame of reference.

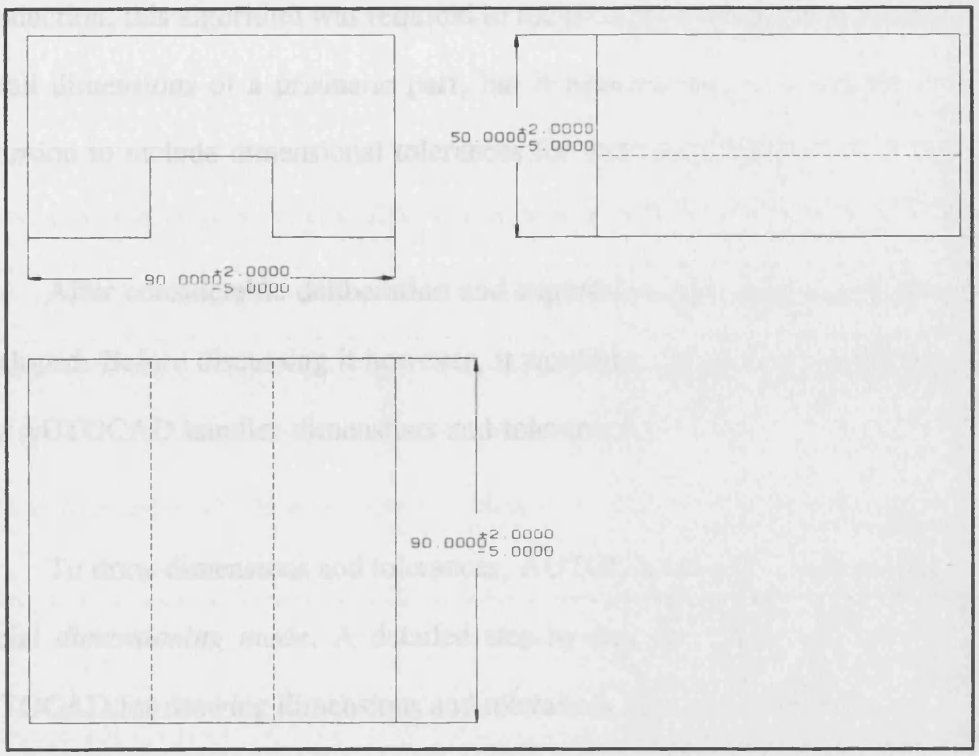


Figure 7.16 A typically overall dimensioned component

This involved initially the creation of a new drawing layer (called DIM for DIMensional) where all the dimensioning entities would reside. However, it was soon realised that the (future) need for accurate individual feature tolerance allocation, necessitated the expansion of this layer into three distinct layers, one for each view of the drawing. The new layers were thus named DIMF (DIMensional Front), DIMP (DIMensional Plan) and DIME (DIMensional End), and it was stipulated that each of these layers should include only the dimensional entities pertinent to a particular view.

Once the basic framework for drawing dimensional tolerances was in place, an expandable algorithm to extract them was developed. As mentioned in the

introduction, this algorithm was required to focus on identifying the tolerances of the overall dimensions of a prismatic part, but it also needed to be capable of future expansion to include dimensional tolerances for individual features.

After considerable deliberation and experimentation, such an algorithm was developed. Before discussing it however, it would be useful at this point to mention how AUTOCAD handles dimensions and tolerances.

To draw dimensions and tolerances, AUTOCAD must first be switched to its special *dimensioning mode*. A detailed step-by-step description of how to set up AUTOCAD for drawing dimensions and tolerances suitable for BUFIP can be found in **Appendix A**. Once in its dimensioning mode, AUTOCAD can inset dimensions into a drawing in two ways. The first of these methods is referred to as a *normal* dimension; the second as *associative*. In the first method, the individual entities that make up the dimension are inserted as individual entities in the ENTITIES section of the DXF file. Hence, the dimension text is a text entity, the dimension lines are line entities etc. It should be noted that with toleranced text, if the upper and lower tolerance limits are the same, then a single combined dimension text entity appears, otherwise three distinct text entities are created (one with the nominal dimension, and two others with the different tolerance limits).

With *associative* dimensioning on the other hand (AUTOCAD's default method), the individual entities that make up a dimension are grouped and treated collectively in "blocks", that appear in the BLOCKS section of the DXF file. For reasons of consistency and compatibility with the programs developed so far, it was

decided to use the *normal* dimensioning mode, since its results appear in the ENTITIES section of the DXF file, already scanned by BUCADIP.

AUTOCAD also uses several different types of dimensions, including linear (horizontal and vertical), radial, angular etc. For the purposes of this research, only linear dimensions have been used. Several toggled system variables are used by AUTOCAD, for controlling the way dimensions are drawn and what they look like. Each of these system variables can be changed if required, but for BUFIP they are left in their default positions.

In a similar manner, other system variables enable the tolerancing information display and type, as well as setting of the upper and lower tolerance limit values. It should be noted that these affect all tolerances drawn thereafter, and have to be reset every time different limits are needed for different features. As mentioned previously, **Appendix A** provides step-by-step guidelines for setting up AUTOCAD and the setting of variables for effective tolerance allocation.

It became evident that a method had to be found that was able to correlate the tolerancing text entities found in the DIM layers to the appropriate edge and feature entities found in the view layers of a drawing. Following testing of several sample components, it was shown that a correlation method could be based on the location and coordinates of the dimensional arrows, and in particular their tips. Consider for example the dimensioning representation of a component's horizontal edge, shown in **Figure 7.17** below:

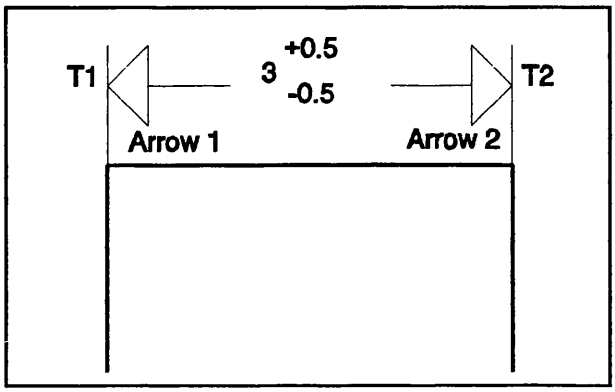


Figure 7.17 Dimensioning representation of a component's horizontal edge

Figure 7.17 shows that the X coordinate of the tip T1 of arrow 1, coincides exactly with the X coordinate of the start of the edge. Similarly, the X coordinate of the tip T2 of arrow 2 coincides exactly with the X coordinate of the end of the edge. Since the tolerancing text entries follow the arrow entity entries in the DXF file format, it can be concluded that they can be referred to where the arrow tips "point" (in this case the edge of the component).

Now consider the dimensioning representation of a component's vertical edge, shown in **Figure 7.18** below:

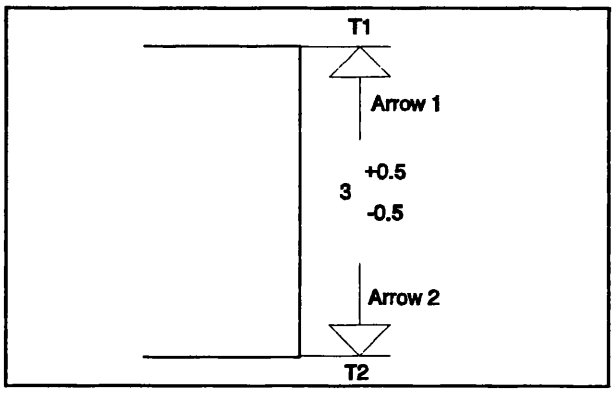


Figure 7.18 Dimensioning representation of a component's vertical edge

In this case the Y coordinate of the tip T1 of arrow 1 coincides with the Y coordinate of the start of the edge (remember edges are sorted left-to-right and top-to-bottom). Similarly the Y coordinate of the tip T2 of arrow 2 coincides with the Y coordinate of the end of the edge, and again the ensuing tolerancing text entries can be allocated to where the arrow tips "point".

The system needed to distinguish between arrows "pointing" along the horizontal plane (so that it can use their X tip coordinates as in **Figure 7.17**) and arrows "pointing" along the vertical plane (so that it can use their Y tip coordinates as in **Figure 7.18**). A solution to this problem was found by remembering that dimensional arrows come in pairs; then if the X coordinates of the tips of arrows 1 and 2 are the same, this pair of arrows "points" along the vertical plane (as in **Figure 7.18**), otherwise it "points" along the horizontal plane.

This methodology of identifying where the tips of dimensional arrows "point", and allocating the ensuing tolerance text entries appropriately, can be expanded to cover individual features on a component. Consider for example a dimensioned through slot, such as the one shown in **Figure 7.19** below:

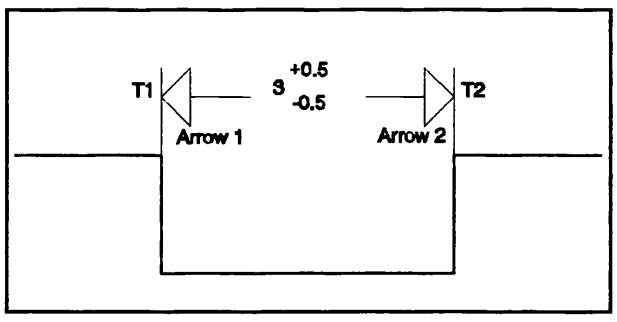


Figure 7.19 Dimensioning representation of a through slot

In this case, the X coordinate of the tip T1 of arrow 1 corresponds to the X coordinate of the start of the slot (already provided by BUFIP), while the X coordinate of the tip T2 of arrow 2 corresponds to the X coordinate of the end of the slot (readily determined by BUFIP). The ensuing tolerance text entries could thus be effectively allocated. In a similar manner, other features with linear dimensions (such as steps, pockets and hole depths) could be processed successfully. For hole diameters however, the algorithm might need some modifications to cope with dimensioning for circular entities.

Focusing again on the problem of allocating tolerances to the overall dimensions of a component, it was decided in this case to adopt the standard engineering practice of describing the component's sides in terms of X, Y and Z coordinates. In this way, a more conventional expression for the component's required block shape envelope, as well as for the tolerances of each dimension could be obtained. A convention was thus defined for the allocation of dimensioning and tolerancing text on the appropriate sides of a part. This was as shown in **Figure 7.20** below.

Using this convention, a tolerance found as belonging to, say, side 0 of the plan view, represents the X dimension tolerance of the component, and so on. In this way, tolerancing text found at any side of any view can be effectively allocated, using the previously described algorithm, to the X, Y or Z dimensions of the part (essentially its length, width or depth). This makes it easier to visualise the required tolerances for the part, as well as the volume of its block shape.

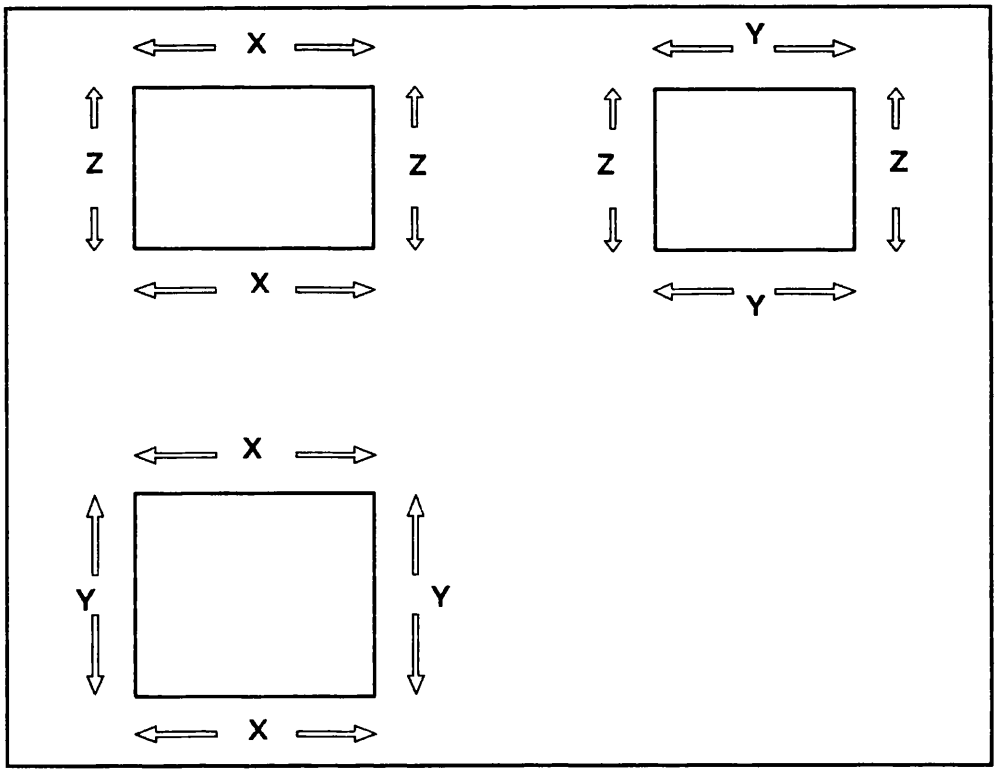


Figure 7.20 Convention used for the allocation of dimensioning text

The maximum possible block size is the minimum required raw material block shape envelope for the component and this can be calculated as follows:

Required raw material block shape envelope=

(required total length) * (required total width) * (required total depth) =

(X dimension + upper X tolerance) * (Y dimension + upper Y tolerance) *

(Z dimension + upper Z tolerance).

Now let us consider in detail how tolerances are applied in programming code. Tolerance extraction and manipulation is performed by **tolrance.c**. This takes as its input the stored and structured text and arrow tip entities, already extracted by **props.c** from the interpreted data file (note that BUCADIP, when interpreting a DXF file, was upgraded to automatically determine an arrow's tip coordinates and

record only them in the interpreted file).

Tolrance.c is a relatively small program compared with the others, and consists of only one function: *Tolerances*. This is responsible for identifying and extracting tolerance limits, allocating them to the appropriate component X, Y or Z dimensions, reporting results, and calculating and reporting the block shape envelope.

Function *Tolerances* starts by examining the arrow tip entities for each dimensional layer. According to the previous discussion, there will be one (for identical upper and lower tolerance limits) or three (for different upper and lower tolerance limits) associated text entries for each pair of arrow tips encountered. Starting with the first encountered pair of arrow tips, the nominal dimension value, as well as the values for the upper and lower tolerance limits are extracted from the ensuing text entities and stored in appropriate variables.

The system now implements the allocation algorithm. Depending on the dimensional layer on which the arrow tips were found, their location and their orientation (along the horizontal or vertical planes as mentioned previously), the stored nominal dimension and tolerance values are reassigned to the appropriate X, Y or Z dimensions of the prismatic part, according to the convention shown in **Figure 7.20.**

The process is then repeated for the next encountered pair of arrow tips, and so on until all arrow tips and their associated text entries are processed. In the case where the system cannot detect the presence of any arrow tips or associated text

entries, a default is next implemented. This assigns the maximum values of the edges from each layer of the drawing to the component's nominal X, Y or Z dimensions according to the previously described convention. Thus, even if the part drawing does not contain any dimensioning or tolerancing information, its overall dimensions (and hence its block shape envelope) can still be determined.

The system next reports its findings. The produced report, recorded in the output file, has the following format:

"Length X = (length)

Tolerance X+ = (upper X tolerance)

Tolerance X- = (lower X tolerance)

Width Y = (width)

Tolerance Y+ = (upper Y tolerance)

Tolerance Y- = (lower Y tolerance)

Depth Z = (depth)

Tolerance Z+ = (upper Z tolerance)

Tolerance Z- = (lower Z tolerance) "

where

(length)=the discovered nominal X dimension of the component (in mm).

(upper X tolerance)=the upper tolerance limit for the X dimension (in mm).

(lower X tolerance)=the lower tolerance limit for the X dimension (in mm).

(width)=the discovered nominal Y dimension of the component (in mm).

(upper Y tolerance)=the upper tolerance limit for the Y dimension (in mm).

(lower Y tolerance)=the lower tolerance limit for the Y dimension (in mm).

(depth)=the discovered nominal Z dimension of the component (in mm).

(upper Z tolerance)=the upper tolerance limit for the Z dimension (in mm).

(lower Z tolerance)=the lower tolerance limit for the Z dimension (in mm).

The required block shape envelope calculations and report follows immediately afterwards. This is recorded in the following form:

"Block Shape Envelope = (total length) * (total width) * (total depth)"

where

(total length)=the nominal X dimension + the upper X tolerance limit (in mm).

(total width)=the nominal Y dimension + the upper Y tolerance limit (in mm).

(total depth)=the nominal Z dimension + the upper Z tolerance limit (in mm).

A flowchart of the general operation sequence of **tolrance.c** can be seen in **Figure 7.21.**

7.2.3 Some thoughts on geometric tolerance and surface roughness allocation

Geometric tolerancing, as well as surface roughness values, are not directly supported by AUTOCAD. Hence the special symbols denoting their presence (defined in BS308-1972) are not readily available either. They could however be created from simple entities (lines, circles, arcs etc), and then grouped together in "blocks" (one block per symbol), which can be saved separately, and used independently in various drawings. Now, if these special symbols, along with their numerical text values, are drawn in separate layers of the drawing (perhaps a geometric tolerance layer and a surface roughness layer), BUFIP could be able to extract them.

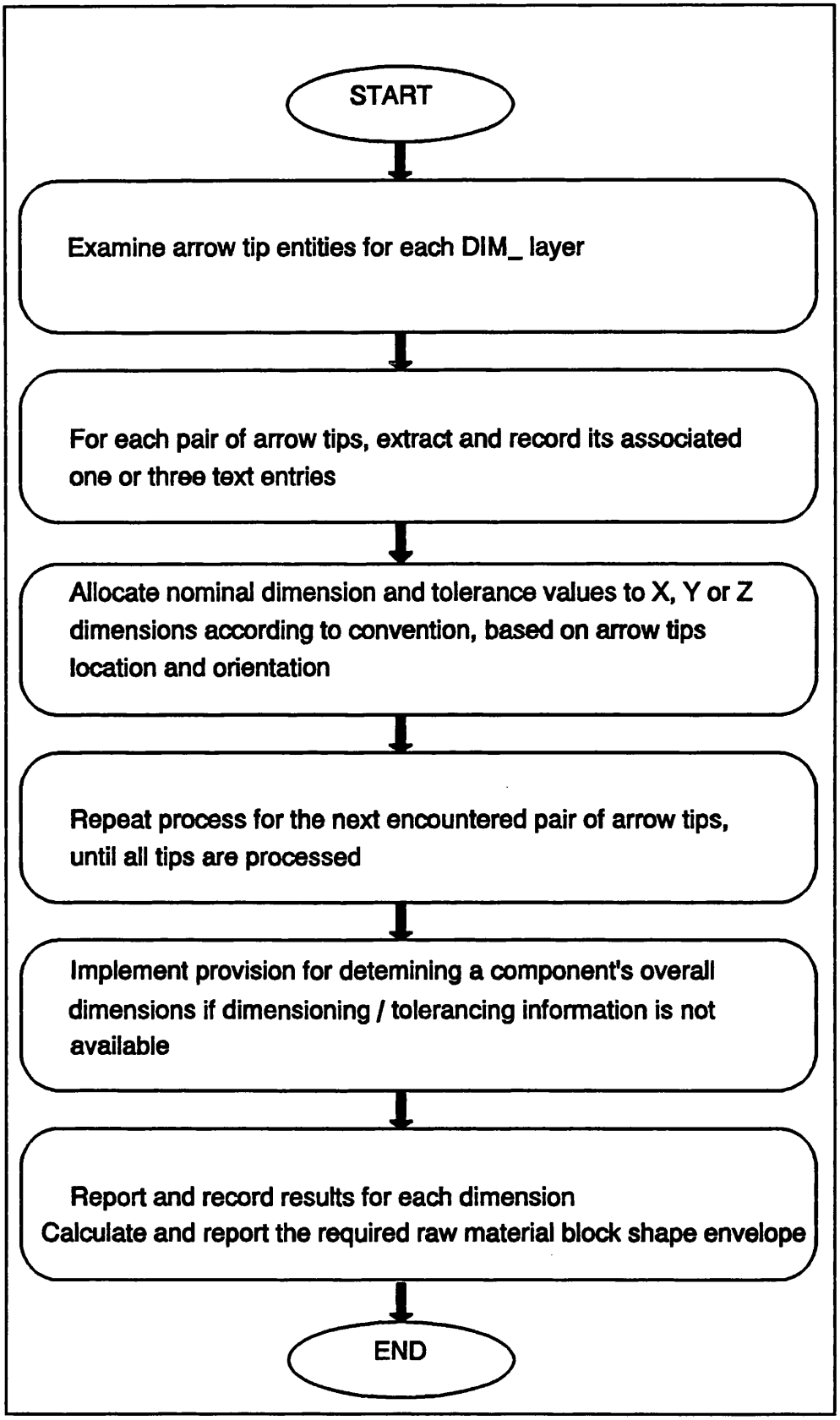


Figure 7.21 General operation sequence flowchart for **tolrance.c**

BUCADIP would have to be modified first, to cope with the "block" entries. This means upgrading its scanning capability to include the BLOCKS section of the DXF file, so that the block definitions and their relation to the ENTITIES section can be established and interpreted appropriately.

Next, new extra modules for BUFIP would have to be coded and "bolted" on to the system. These would include a module for dealing with geometric tolerances, and one for dealing with surface roughness allocation. These modules would require functions to be developed specifically for this task. It is envisaged that such algorithms could be based on a variation of the tolerance allocation algorithm described previously. For example, instead of looking for arrow tip coordinates and correlating them, the algorithms could look for the special block symbol's coordinates and use them as a basis for correlation. Their associated text values could then be assigned to appropriate surfaces on the component.

Apart from the creation of new modules, some core modules of BUFIP would also have to be upgraded to cope with the processing of new extra information. These include **main.c** (from where the new modules would be activated), **props.c** (for dealing with block symbol properties), as well as the error reporting system (**msg.c** and **msgtable.c**) for detecting and reporting new kinds of errors liable to result with the new environment. In the case of component edges requiring extra finishing operations to obtain a certain surface roughness value, an allowance on the block shape envelope dimensions could be added to the total length of the respective surface for machining operations.

CHAPTER 8

BUFIP IN ACTION

8.1 Introduction

Having discussed the theoretical and programming aspects of the BUFIP system, this chapter demonstrates how it actually performs in practice. For this purpose, three characteristic prismatic parts (amongst the many tested) have been chosen to illustrate the various facets of BUFIP's operation. In addition, since the system was envisaged to eventually be able to interface directly with a CAPP package, section 8.6 describes in more detail the framework for such a connection, using a particular application example: Rustom's BEPPS-GSCAPP software (analysed in detail in **Chapter 2**).

Each sample part shown below includes a combination of various features (flat and cylindrical) on various views of its drawing. Additionally, some features are "hidden", or integrated with each other. Dimensioning and tolerancing information (with different tolerance values) is also included with each component to deduce its overall dimensions.

The prismatic parts are presented in the following sections. Each section consists of the drawing of a sample part, along with a detailed listing produced by BUFIP, of the features that it identified. The drawings are shown in 1:1 scale, so that the results from BUFIP can be easily verified using a rule. It should be reminded at this point that the custom coordinate system is implemented (as discussed in

Chapter 5), therefore each layer's origin (0,0) point is at the bottom left hand corner of each view of the drawing.

A brief description of each prismatic component follows below:

Prismatic component 1 includes a hidden closed pocket, and a stepped counterbored, conical-ended hole on its plan view. A simple step and simple slot are also included on the front view. This demonstrates BUFIP's ability to deal with hidden pockets and complex holes.

Prismatic component 2 includes a hidden thread and open pocket on its front view, with multiple slots and multiple steps on its plan view. It should be noted that the dimensioning information is also placed in different positions. This component demonstrated BUFIP's ability to deal with multiple slots and multiple steps as well as threads.

Prismatic component 3 includes all its features on the end view. A simple conical-ended hole, with a hidden open pocket are shown, but the most notable feature is a slot integrated with two steps. Again, dimensioning information is placed in different positions. This component demonstrates BUFIP's ability to deal with an integrated combination of slots and steps, as well as with many different features on the same layer.

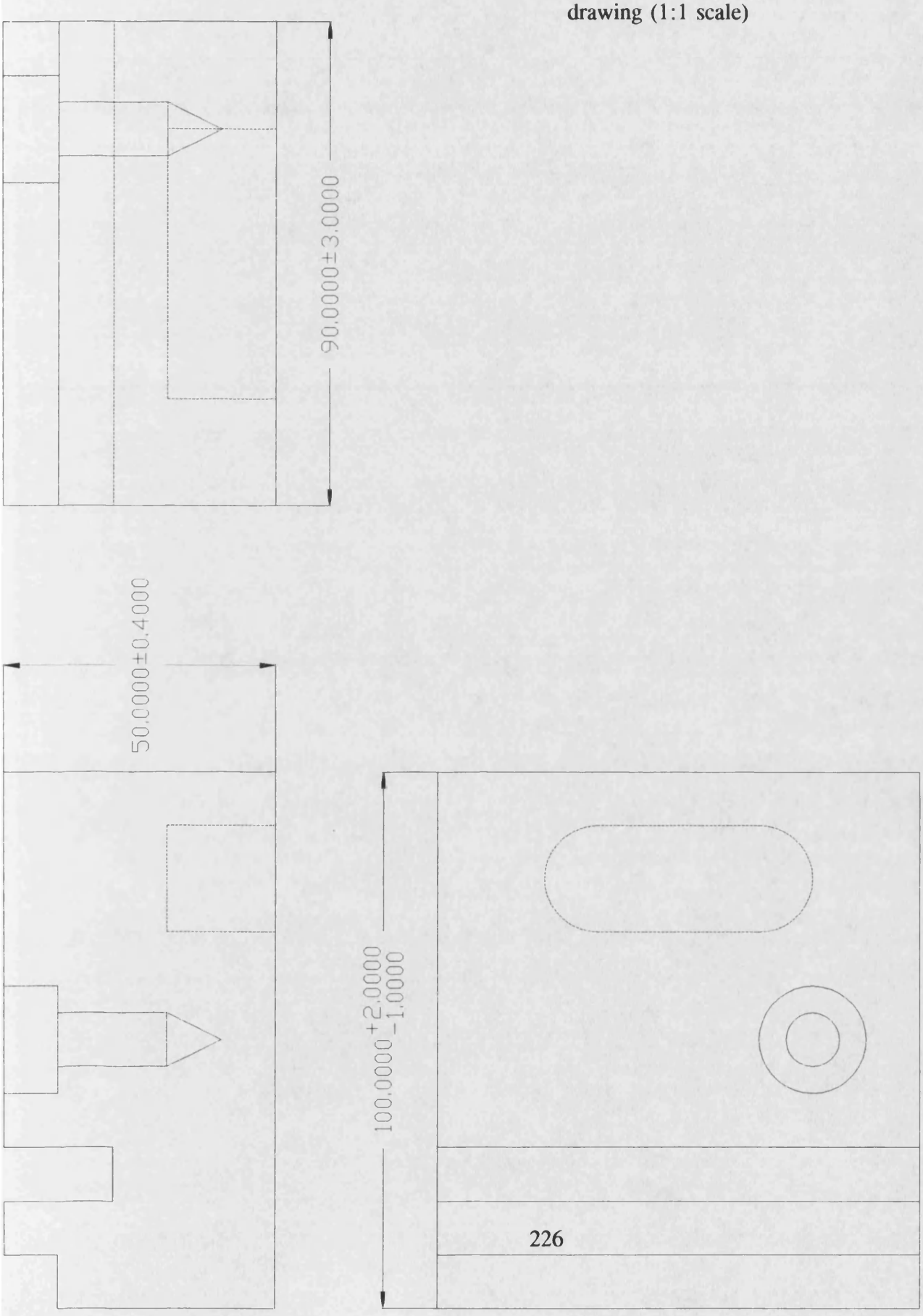
8.2 BUFIP Feature List File and Component 1 Drawing

```
North slot found on Layer 0 Side 0: (X = 20.000000 Y = 50.000000)
        Distance from Edge = 20.000000, Width = 10.000000, Depth = 20.000000
Step found on Layer 0 Corner 0:
        X = 10.000000, Width = 10.000000, Depth = 10.000000
Stepped Conical Ended Hole found on layer 1
        X Centre is 50.000000, Y Centre is 20.000000
        Diameter is 10.000000, Depth is 20.000000
        Step type is Counterbore (Diameter is 20.000000, Depth is 10.000000)
Hidden Closed Pocket on Layer 1
        Width = 50.000000, Length = 20.000000
        Radius = 10.000000, Depth = 20.000000
        Arc 0: X Centre = 80.000000, Y Centre = 60.000000
        Arc 1: X Centre = 80.000000, Y Centre = 30.000000

Length X = 100.000000
        Tolerance X+ = +2.000000
        Tolerance X- = -1.000000
Width Y = 90.000000
        Tolerance Y+ = +3.000000
        Tolerance Y- = -3.000000
Depth Z = 50.000000
        Tolerance Z+ = +0.400000
        Tolerance Z- = -0.400000

Block Shape Envelope = (102.000000) * (93.000000) * (50.400000)
```

Figure 8.1 Prismatic component 1 drawing (1:1 scale)



8.3 BUFIP Feature List File and Component 2 Drawing

```
Hidden Threaded Flat Bottomed Hole found on layer 0
        X Centre is 60.000000, Y Centre is 30.000000
        Diameter is 10.000000, Depth is 30.000000
Open Pocket on Layer 0
        Width = 10.000000, Length = 25.000000
        Radius = 5.000000, Depth = 15.000000
        Arc 0: X Centre = 70.000000, Y Centre = 5.000000
West slot found on Layer 1 Side 3: (X = 0.000000 Y = 50.000000)
        Distance from Edge = 10.000000, Width = 30.000000, Depth = 10.000000
West slot found on Layer 1 Side 3: (X = 0.000000 Y = 50.000000)
        Distance from Edge = 20.000000, Width = 20.000000, Depth = 10.000000
West slot found on Layer 1 Side 3: (X = 0.000000 Y = 50.000000)
        Distance from Edge = 30.000000, Width = 10.000000, Depth = 10.000000
Step found on Layer 1 Corner 3:
        X = 10.000000, Width = 10.000000, Depth = 40.000000
Step found on Layer 1 Corner 3:
        X = 20.000000, Width = 10.000000, Depth = 20.000000
Step found on Layer 1 Corner 3:
        X = 40.000000, Width = 20.000000, Depth = 10.000000


Length X = 90.000000
        Tolerance X+ = +3.000000
        Tolerance X- = -3.000000
Width Y = 90.000000
        Tolerance Y+ = +0.500000
        Tolerance Y- = -0.500000
```

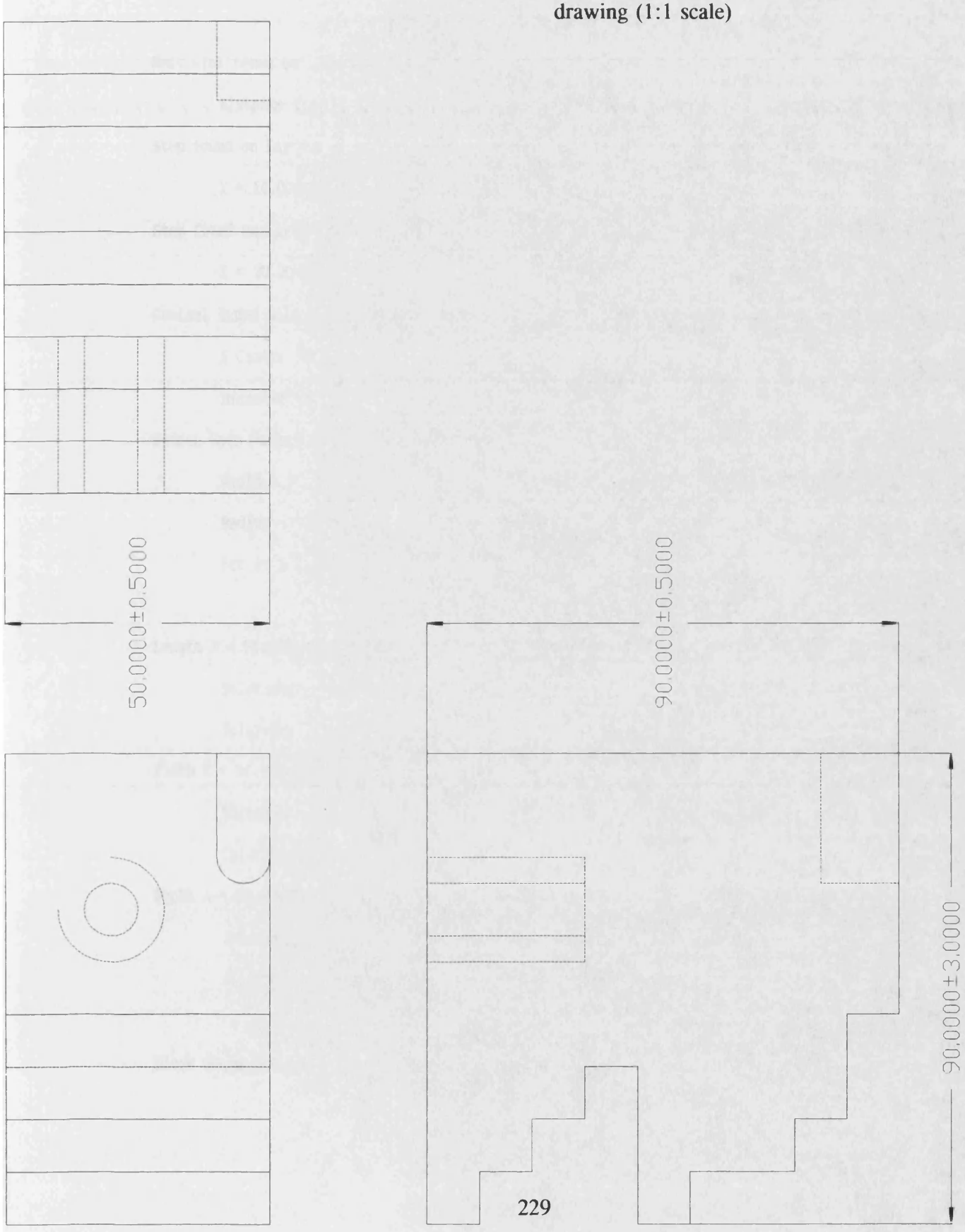
Depth Z = 50.000000

Tolerance Z+ = +0.500000

Tolerance Z- = -0.500000

Block Shape Envelope = (93.000000) * (90.500000) * (50.500000)

Figure 8.2 Prismatic component 2 drawing (1:1 scale)



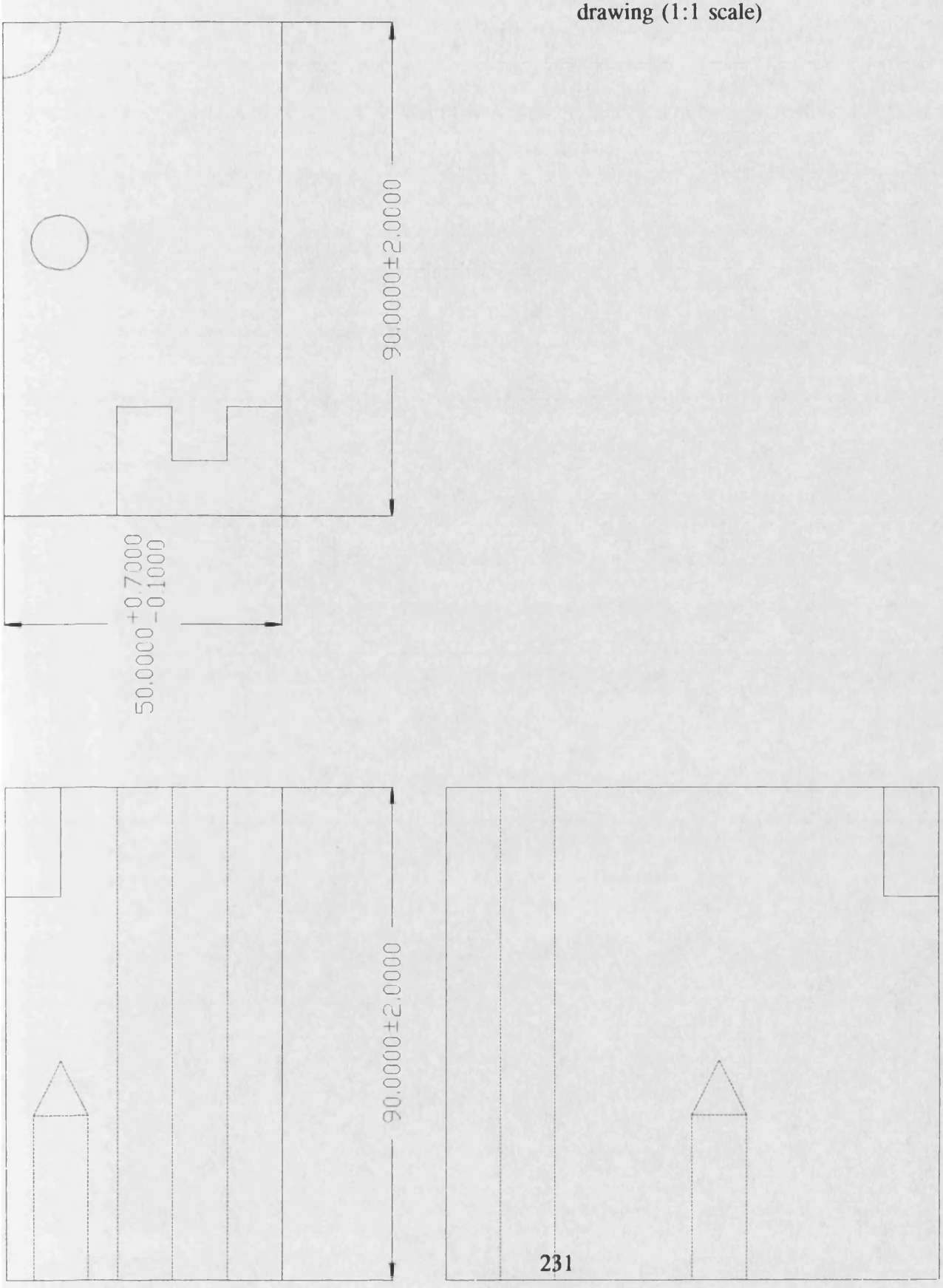
8.4 BUFIP Feature List File and Component 3 Drawing

```
West slot found on Layer 2 Side 2: (X = 10.000000 Y = 20.000000)
        Distance from Edge = 20.000000, Width = 10.000000, Depth = 10.000000
Step found on Layer 2 Corner 3:
        X = 10.000000, Width = 10.000000, Depth = 30.000000
Step found on Layer 2 Corner 3:
        X = 20.000000, Width = 10.000000, Depth = 10.000000
Conical Ended Hole found on layer 2
        X Centre is 50.000000, Y Centre is 40.000000
        Diameter is 10.000000, Depth is 30.000000
Hidden Open Pocket on Layer 2
        Width = 10.000000, Length = 10.000000
        Radius = 10.000000, Depth = 20.000000
        Arc 0: X Centre = 90.000000, Y Centre = 50.000000


Length X = 90.000000
        Tolerance X+ = +2.000000
        Tolerance X- = -2.000000
Width Y = 90.000000
        Tolerance Y+ = +2.000000
        Tolerance Y- = -2.000000
Depth Z = 50.000000
        Tolerance Z+ = +0.700000
        Tolerance Z- = -0.100000


Block Shape Envelope = (92.000000) * (92.000000) * (50.700000)
```

Figure 8.3 Prismatic component 3 drawing (1:1 scale)



8.5 Concluding Remarks on the Results

As it can be seen from the previous sections, the system is very successful at identifying complex profiles with a multitude of different features located on different sides of a component. Various tolerance values, located in different views and sides of a component are also accurately extracted. This indicates the flexibility of the system, as well as its suitability for resolving a wide variety of prismatic parts in realistic situations.

It should be noted that the order in which features are extracted was not optimised in any way. The system simply executes the feature identification modules and presents the results in the sequence discussed in **Chapter 3**. Future research could focus on an optimisation strategy for reordering the presentation of the results (or changing their output format) for more effective subsequent process planning.

As it was mentioned previously, the components shown in these examples are just a small sample of the total number of components tested. The system was extensively validated over a wide range of diverse prismatic parts (within the research work limitations mentioned in **Chapter 1**) and proved to be very robust. Testing also helped to fine tune some aspects of the system, as well as to explore and document the system's current limitations.

8.6 Interfacing BUFIP with BEPPS-GSCAPP

In order to demonstrate, in practical terms, that the feature based format of BUFIP's output files has the potential for driving directly a CAPP system, it was decided to investigate in more detail how this could be accomplished using Rustom's BEPPS-GSCAPP software [Rustom 1992] as a particular application example.

Figures 8.4 and **8.5**, taken after Rustom [1992] show the range of features and their particular characteristics that BEPPS-GSCAPP requires to produce detailed process plans for prismatic parts. Indeed Rustom states in his thesis that if a CAD interface program could provide an output file with this feature information, it could support BEPPS-GSCAPP. It can be easily observed, from the typical sample prismatic components presented in the previous sections, that BUFIP's output format file contains all this information and more (such as hole and pocket locations for example). The only information not directly provided by BUFIP pertains to individual feature tolerances and surface roughness requirements which, as mentioned previously, is outside the scope of this research work.

Comparing the two systems, the slots, pockets and holes parameters required by GSCAPP are identical to the ones provided by BUFIP, while for steps the defined "length" of the step according to Rustom is termed its width by BUFIP (easily modified). Countersunk holes for input to GSCAPP require the included countersink angle which could be calculated as follows:

Included countersink angle C = 180deg - (2 * A) where

angle A = Depth of countersink step / (Radius of large hole - Radius of small hole).

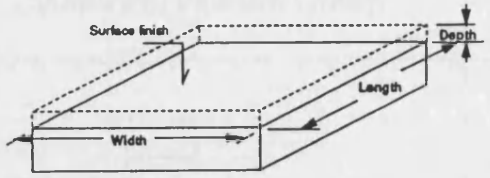
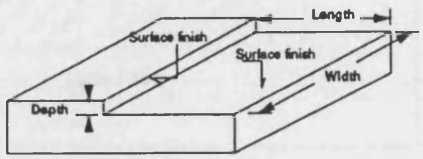
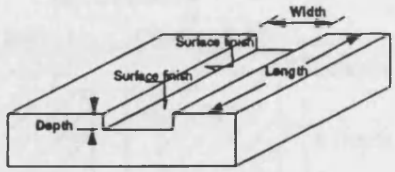
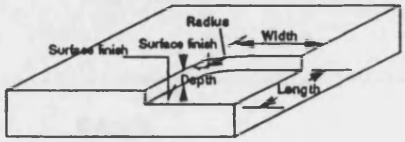
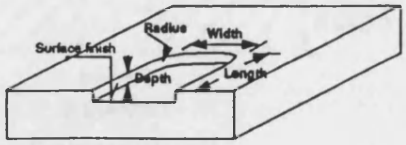
Flat Feature Group	Input Data
 Flat Surface	Length. Face Roughness. Width. Depth.
 Step Face	Length. Face Roughness. Width. Side Roughness. Depth.
 Slot	Length. Face Roughness. Width. Side Roughness. Depth.
 Open Pocket	Length. Face Roughness. Width. Side Roughness. Depth. Radius.
 Side Pocket	Length. Face Roughness. Width. Side Roughness. Depth. Radius.

Figure 8.4 Flat feature input data required by BEPPS-GSCAPP (from [Rustom 1992])

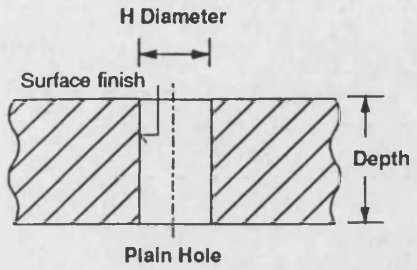
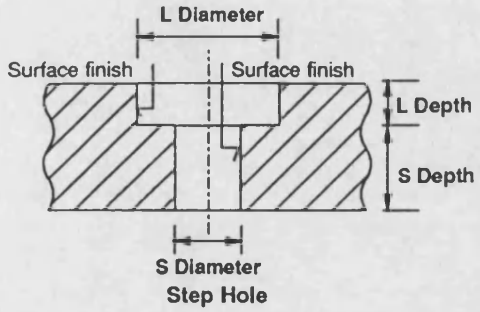
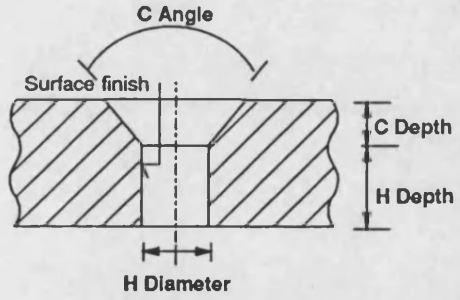
Cylindrical Feature Group	Input Data
	Hole Diameter. Hole Depth. Hole Tolerence. Surface Finish.
	Large Hole Diameter. Large Hole Depth. Large Hole Tolerence. Large Hole Surface Finish. Small Hole Diameter. Small Hole Depth. Small Hole Tolerence. Small Hole Surface Finish.
	Countersink Angle. Countersink Depth. Hole Diameter. Hole Depth. Hole Tolerence. Surface Finish.

Figure 8.5 Cylindrical feature input data required by BEPPS-GSCAPP (from [Rustom 1992])

Also, since BUFIP and BEPPS-GSCAPP use different classification schemes for describing a component's planes and edges, some sort of conversion process would also be required to map information from one system to the other. The following highlights the various aspects of such a conversion framework by comparing the two systems' operational concepts:

1. Overall dimensions of a component:

In Rustom's system the planner has to specify the largest dimension of a component as its *length* (assigned as X), its medium dimension as its *width* (assigned as Y) and its smallest dimension as its *depth* (assigned as Z).

BUFIP is much more flexible, since any dimension of a component can be drawn in any view, and hence become X, Y or Z according to the layout shown in **Figure 7.20**. However, in order to maintain specific compatibility with BEPPS-GSCAPP, the user should ensure that the largest dimension of a component's drawing should be placed along the X-axis (for example sides 0 or 2 of the PLAN view), while its smallest dimension should be placed along the Z-axis (for example sides 1 or 3 of the FRONT view).

2. Edge coding:

In BEPPS-GSCAPP edges are coded in an anti-clockwise direction relative to their plane positions and original X, Y, Z axis positions.

Hence for X they are termed EX0, EX1, EX2, EX3

for Y they are termed EY0, EY1, EY2, EY3

and for Z they are termed EZ0, EZ1, EZ2, EZ3.

BUFIP codes edges numerically in a clockwise direction according to the view layer of the drawing starting from the top edge of the front layer (edge 0, FRONT layer). Z dimensions are not used, while corners are also classified in the same way as edges.

An edge mapping program would therefore be needed, able to convert BUFIP edge codes into BEPPS-GSCAPP edge codes according to the following table:

BUFIP		BEPPS-GSCAPP
FRONT layer	Side 0	EX2
	Side 1	EZ1
	Side 2	EX1
	Side 3	EZ0
"Hidden"	Side 2	EX0
PLAN layer	Side 0	EX3
	Side 1	EY2
	Side 2	EX2
	Side 3	EY1
"Hidden"	Side 1	EY3
END layer	Side 0	EY1
	Side 1	EZ0
	Side 2	EY0
	Side 3	EZ3
"Hidden"	Side 3	EZ2

3. Plane surface coding:

BEPPS-GSCAPP defines 3 datum planes (coded as XD, YD, ZD) formed by edges where the values of X, Y, Z coordinates respectively are equal to zero. Three

"opposite" planes (opposite the datum planes) are also defined, and coded as XO, YO, ZO, where the edge values of X, Y, Z coordinates respectively are equal to Xmax, Ymax, Zmax.

BUFIP has no plane surface coding, but plane surface codes could be built into the conversion program mentioned previously using the following correlations:

BEPPS-GSCAPP plane codes	Respective BUFIP edges - layers
XD	END layer sides 0, 1, 2, 3.
YD	FRONT layer sides 0, 1, 2, 3.
ZD	PLAN layer hidden sides 0, 1, 2, 3.
XO	END layer hidden sides 0, 1, 2, 3.
YO	FRONT layer hidden sides 0, 1, 2, 3.
ZO	PLAN layer sides 0, 1, 2, 3.

4. Component type determination:

When inputting component information into BEPPS-GSCAPP, the planner has to determine a component's "type" and enter it into the system. This can be of Totally Constant, Partially Constant or of Non Constant Cross-Section, depending on the existing flat feature types and locations. Rustom in his thesis provides the rules for determining a component's "type".

BUFIP on the other hand does not provide any such classification for prismatic parts. The conversion program could however, readily deduce a component's type by examining the feature-based BUFIP output file and applying Rustom's rules on the detected flat features and their location information. It could then pass on this information to BEPPS-GSCAPP.

5. Overall shape envelope:

The overall shape envelope of a component is typed into BEPPS-GSCAPP in terms of Length, Width and Depth together with their dimensional tolerances. Roughness requirements for all surface planes are also typed in. Rules are used to combine tolerance and surface roughness requirements in order to determine the appropriate raw material block size.

BUFIP automatically extracts dimensional tolerances and determines the raw material block size. All this information is presented in the output file. Surface roughness requirements, considered to be outside the scope of this research work as mentioned previously, are not supported, and would still need to be input manually.

From the above discussion it can be seen that apart from surface roughness BUFIP provides all the necessary information for directly driving a CAPP system (Rustom's BEPPS-GSCAPP in this particular application example). A conversion program is only required, in order to adapt the feature based component information extracted by BUFIP into an appropriate input format for BEPPS-GSCAPP. This conversion program would have to be "bolted" on BEPPS-GSCAPP's data input functions in order to automate the (currently manual) data input process, by making it capable of directly reading in BUFIP produced component data files.

CHAPTER 9

CONCLUSIONS, RESEARCH ACHIEVEMENTS AND SUGGESTIONS FOR FURTHER WORK

9.1 Conclusions and Research Achievements

The final versions of the BUFIP feature identification system and BUCADIP CAD DXF interpreter are considered to meet the objectives set at the beginning of the research work, and outlined in **Chapter 1**. The primary aim of automating the CAD to CAPP interface for prismatic parts, typically produced in a small to medium batch manufacturing environment, has been effectively achieved. Once a component is drawn according to the guidelines described in earlier chapters (and its DXF file generated), the system requires no other human input for converting entity data into manufacturing features, ready for process planning. However, due to the nature of the research work, some limitations have been imposed on the flexibility and scope of the system (as discussed in **Chapter 1**).

Concluding, the following objectives and research achievements have been accomplished:

- The system has been built with expandability and ease of maintenance in mind, hence a modular structure has been used. Individual modules are organised in a logical order, usually with each module implementing a specific task (for example **holes.c** deals only with identifying holes in prismatic components). Thus the system

readily provides the means for easy updating and upgrading of both the recognition algorithms, and the range of features the system can detect.

- The industry standard DXF file format of a typical three view engineering drawing of a prismatic component is used as the system input. In this way compatibility is retained with both existing engineering practices, as well as with a wide range of CAD systems currently used in industry.

- The results that the system produces are automatically recorded in a data file using the standard ASCII format. This should make them directly usable by an appropriate CAPP system, or easily checked and edited by human operators if needed. The compatibility of the produced results with an existing CAPP system was described using Rustom's BEPPS-GSCAPP software [Rustom 1992] as a particular application example.

- The selection of the IBM-PC compatibles as the platform for developing the system provides a cost effective solution for the small to medium batch manufacturing companies, the envisaged eventual users of the system. However, the selection of the C language for programming means that the system can be easily ported to other platforms if necessary, hence enhancing its flexibility. Also C provides the system with efficient use of computer memory, as well as speed of execution. The software is completely automatic in its operation, requiring just a single command from the MS-DOS prompt to be activated. Hence it is easy and friendly to use even by an unskilled operator.

- The application of the layered view approach for a component CAD engineering drawing, simplifies considerably the interpretation process, since it solves the problem of the same features being duplicated on different views of the drawing. Assigning entities from a particular view to a particular layer is the main requirement the system expects from the user and it is not considered unusually demanding (the practice of layering CAD drawings is widely used in industry).

- The BUCADIP interpreter has been developed to efficiently "filter" the (large) DXF files produced by AUTOCAD (in this case). Such files contain a lot of redundant data not necessarily useful for feature identification or process planning. The resultant interpreted file (in ASCII format) is much smaller, clearer, and can be easily edited manually if required. Descriptive language statements are used for the various entities and their associated parameters (rather than group codes or numbers) thus greatly facilitating editing or cross-referencing the file with the original drawing for verification purposes. The separation of BUCADIP from the main part of BUFIP means that it could be modified to handle other standard CAD file formats (eg IGES) and still produce the same interpreted file format for feature identification. This program is also completely automatic in its operation, requiring just a single command from the MS-DOS prompt to be activated.

- The system employs a comprehensive error detection and reporting system. This enables it to cater for a wide range of malfunctions, and alert the user to their cause, while at the same time attempting to identify features and complete its task without halting. This of course depends on the nature of the error encountered. The error messages are located in separate modules in order to facilitate the addition of extra

error messages in the future.

- The custom coordinate system implemented in BUFIP not only facilitates manual checking of the identified features and their characteristics, but also resembles more closely the actual selection and referencing of datum points for the component in a real machining environment. Thus it was felt that development of this internal classification scheme for the workpiece and its features would also prove beneficial for subsequent effective process planning. However, the definition of this custom coordinate system does not preclude its conversion to another coordinate system with a different point (or points) of origin, if required. This would be straightforward to implement, by simply modifying the existing developed coordinate transformation algorithms.

- The division of the component profile in each view of a drawing into four distinct sides and corners provides much more accurate feature location reporting. However, the existence of ambiguous component profiles prompted the development of an algorithm that was able to decide under various situations how the four component sides are formed. The algorithm was found to be able to handle any component profile (within the research limitations) that it is given.

- The through slot recognition algorithm was progressively developed to cope effectively not only with any highly complex combination of multiple slots, but also with combinations of integrated multiple slots and multiple steps. However, in its latest version, it might occasionally double report the same area. None the less, this was felt to be better than to completely miss certain slots on certain profiles. The

algorithm not only reports the identified slots and their location / orientation, but is also able to extract the characteristics necessary for process planning (width, depth, distance from edge).

- The step identification algorithm was also enhanced using a similar evolutionary process to the through slot one. It can now successfully recognise not only multiple stepped faces located on any corner of any view of the drawing, but also multiple steps integrated with multiple through slots in any combination. Again, detailed characteristics for each step are provided (width, depth, location).

- The pocket recognition algorithm was developed to handle open, side and closed pockets in either normal or "hidden" situations. However, due to the research nature of the work as well as time limitations, pockets were not allowed to overlap or combine, and only "orthogonal" pockets were considered. A crucial element of the identification strategy relies on being able to determine an arc's start and end points coordinates, which are not readily provided. An algorithm was hence developed to solve this problem. The strategy is also based on a developed pocket "logic table", able to determine a pocket's type depending on the number and type of its constituent lines and arcs. Also, for the first time, information from different views of the drawing had to be correlated in order to determine a pocket's depth. The final version of the developed algorithm can resolve not only a pocket's type, status (normal or hidden) and location, but also correctly calculate all its associated dimensions (length, width, depth, radius of curvature).

- The cylindrical features identification algorithm is capable of coping with plain

holes, as well as stepped (counterbored or countersunk) holes and threads. The system can also decide whether a hole goes through the component, or is flat-bottomed or conically ended. As in the case of pockets, combined multiple holes were excluded from this study, with the exception of threads within stepped holes (a common practical occurrence). Also, hole profiles were not allowed to completely and exactly overlap, but the system can cope with various other degrees of overlap.

- The dimensional tolerance allocation algorithm was developed to extract the dimensional tolerances of a prismatic part's overall dimensions. However the system was designed to be expandable to cover a component's individual feature tolerances, and it is shown how this can be achieved. For this purpose, three additional "dimensional" layers were defined for a drawing, one for each view. Each of them contains all the dimensioning and tolerancing information relating to a particular view. Currently the system can only handle linear (horizontal or vertical) dimensioning. A method was developed, which was able to correlate tolerancing text entries found in the dimensional layers to the appropriate edge and feature entities found in the view layers of a drawing. This is based on locating where the tips of dimensional arrows "point". The standard engineering practice of describing the component's sides in terms of X, Y and Z coordinates was adopted for the presentation of these results. A convention was hence defined for the allocation of dimensioning and tolerancing text on the appropriate sides of a part. The system also determines and reports the component's required raw material block shape envelope (the minimum block of raw material required to machine the part) based on the dimensioning and tolerancing information given. It is also capable of determining the block shape envelope even if no dimensioning or tolerancing data is available. The

system cannot currently cope with geometric tolerances or surface roughness information, but some suggestions have been put forward on how these can be extracted.

- The system and its associated algorithms were exhaustively tested on a wide variety of prismatic component drawings. This enabled the enhancement and fine-tuning of algorithms to cope with diverse situations. It also helped to explore and document the limits of the current programs. The testing process, which took a considerable amount of research time, was therefore crucial in making the system as robust as possible, as well as expanding its capabilities. The robustness and flexibility of the system indicate that it has the potential to be used commercially, particularly if the package is enhanced and a graphical user interface (eg a Windows application) is employed.

9.2 Recommendations for Further Work

The BUFIP system and BUCADIP CAD interpreter form a novel method for automating the effective transfer of data from CAD to CAPP for prismatic components. The research nature of the system however places certain boundaries on its capabilities and logic. Despite this, the developed system offers considerable potential for expansion and enhancement. In fact it was designed from the outset to be expandable, hence its modular structure. New algorithms in new modules can simply be "bolted on" to the current ones, thus increasing the system's power and flexibility. Some recommendations for enhancing existing system components, as

well as for exploiting further areas of research that have been opened up by this work are discussed below, together with possible courses of action for certain areas:

- Upgrading of the through slot algorithm to stop the occasional double reporting of certain slots. This could possibly be achieved by calculating the areas that slots "cover", and discounting any area (and hence the slots that cover it) already covered by another slot.

- Enhancement of the pocket identification algorithm to include non-"orthogonal" pockets. This prerequisites further development of the algorithm that calculates an arc's start and end point coordinates, so that it can cope with any arc and not just orthogonal ones. This could be achieved by applying certain geometry rules. The pocket "logic table" could also be updated to handle more arcs with included angles other than 90 or 180 degs. Pockets with non-homogeneous depths (perhaps stepped or tapered), pockets that overlap or combine in various degrees, as well as pockets having their centreline outside the component block are two more areas that further research work can focus on.

- Updating of the hole recognition program to cover combined multiple holes, as well as exactly overlapping hole profiles. This could perhaps be accomplished by examining the profile of a hole in a second view of the drawing if a conflict in the main examination view is detected.

- Upgrading of the tolerance allocation program to handle tolerancing information for a component's individual features. A suggested method for doing this is discussed in

section 7.2.2.

- Introduction of geometric tolerance and surface roughness allocation processing to the system. Some thoughts on how this can be realised are discussed in section 7.2.3.

- Enhancement of the capabilities of the system to include effective identification of components with inclined faces and features, or of components not necessarily machined from a solid block of raw material (for example castings or forgings).

- Incorporation into the system of a decision-making ability for automatic selection and reporting of a component's recommended clamping surface and datum points, as well as for optimum reordering of the identified features (for more effective subsequent process planning).

- Upgrading of the system to effectively handle "protruding" features on prismatic components (eg "islands").

- Automatic interfacing of the system with appropriate CAPP software, or with another CAD standard format. This might necessitate the porting of BUFIP and BUCADIP to another operating platform (perhaps a UNIX-based workstation), something easily achieved since the software is written in standard C language.

REFERENCES

Abdalla H. and Ikonopisov A., *Automated Feature Recognition for Process Planning,* Proceedings of the Ninth National Conference on Manufacturing Research, pp32-36, Bath, 1993.

Ajmnal A. and Zhang H.G., *State of the art review of CAPP: its impact and its application in CIM,* Proceedings of the Tenth National Conference on Manufacturing Research, Loughborough, 1994.

Ali Z. and Motavalli S., *Requirements from a new generation of CAPP software,* Proceedings of the Industrial Engineering Research Conference, pp797-801, Nocross, GA, USA, 1993.

Alting L. and Zhang H.C., *Computer Aided Process Planning- The State of the Art Survey,* International Journal of Production Research, Vol. 27, No. 4, pp. 553-585, 1989.

Armstrong G.T., Carey G.C., and de Pennington A., *Numerical code generation from a geometric modelling system,* Solid Modelling by Computers: from Theory to Application, Plenum Press, USA, 1984.

Autodesk Inc., *AUTOCAD User Reference,* 1986.

Awadh B., Sepehri N. and Hawaleshka O., *Computer-aided process planning model based on genetic algorithms,* Computers & Operations Research, Vol. 22, No. 8, pp841-856, Oct 1995.

Barkovy B.E and Zdeblick W.J., *A Knowledge Based System for Machining Operation Planning,* Autofact 6, Computer and Automated Systems of SME, Conference Proceedings, Anaheim, California, USA, 10/1984.

Bedworth D.D, Henderson M.R., and Wolfe P.M., *Computer Integrated Design*

and Manufacturing, McGraw-Hill, 1991.

Black R., *Design and Manufacture - an integrated approach,* Mc Millan, 1996.

Bobrow J.E., *NC machine tool path generation from CSG part representations,* Computer Aided Design, Vol. 17, No. 2, pp69-76, 1985.

Bond A.H., Melkanoff M.A., Ahmed S., Zia, Chang K.J., Kim D.H. and Soetarman B., *Automatic Extraction of Geometric Features from CAD Models,* Intelligent Manufacturing Systems II, pp143-160, Elsevier Science Publishers BV, Amsterdam, 1988.

Borland International Inc., *Turbo C++ Programmer's Guide,* 2nd Edition, Borland International Inc, 1991.

Bronsvoort W.F. and Jansen F.W., *Feature modelling and conversion - key concepts to concurrent engineering,* Computers in Industry, Vol. 21, No. 1, pp61-86, 1993.

Brooks S.L, Hummel K.E and Wolf M.L *XCUT: A Rule Based Expert System for the Automated Process Planning of Machine Parts,* Symposium of Integrated and Intelligent Manufacturing, pp181-194, ASME Winter Annual Meeting, Boston, USA, 12/1987.

Brown K.N., McMahon C.A., and Sims-Williams J.H., *Features aka the semantics of a formal language of manufacturing,* Research in Engineering Design, Vol. 7, No. 3, pp151-172, 1995.

Butterfield W.R., Green M.K., Scott D.C. and Stoker W.J., *Part features for process planning,* CAM-I C-85-PPP-03, 1985.

CADCENTRE, *C - PLAN,* Cambridge, United Kingdom.

CAM-I, *Volume decomposition algorithm,* Final Report, General Dynamics

Corporation, R-85-ANC01, Arlington, Texas, 1985.

Case. K. and Acar B.S., *The manufacturing features approach to design and its impact on the learning and use of Computer Aided Design systems,* Contemporary Ergonomics, Taylor and Francis, London, 1989.

Case K. and Gao J.X., *Feature technology: an overview,* International Journal of Computer Integrated Manufacture, 1/1993.

Chang T.C., *Expert Process Planning for Manufacturing,* Addison-Wesley Publishing Company, 1990.

Chang T.C. and Wysk R.A., *An Introduction to Automated Process Planning Systems,* Prentice-Hall Inc., Englewood, Cliffs, New Jersey, USA, 1985.

Choi B.K., Barash M.M., and Anderson D., *Automatic recognition of machined surfaces from a 3D solid model,* Computer Aided Design, Vol. 16, No. 2, pp81-89, 1984.

Chryssolouris G., *Manufacturing systems: theory and practice,* Springer-Verlag, NY, 1992.

Chuang S.H. and Henderson M.R., *Three-dimensional shape pattern recognition and vertex classification and vertex-edge graphs,* Computer Aided Design, Vol. 22, No. 6, 8/1990.

Cutkosky M., Tenenbaum J. and Miller D., *Features in process-based design,* Proceedings of the ASME Computers In Engineering Conference, San Franscisco, USA, 8/1988.

Czajkiewicz Z.J. and Wielicki T.R., *CIM - a journey to manufacturing excellence,* Computers and Industrial Engineering, Vol. 27, No. 1-4, pp91-93, 1994.

Danner W.F., *A proposed framework for STEP (Standard for the Exchange of*

Product Data), NIS-MR 90-4295, National Institute of Standards and Technology, US Department of Commerce, 4/1990.

Davies B.J and Darbyshire I.L *The use of expert systems in process planning,* CIRP Annals, 33(1), 1984.

De Martino T., Falcidieno B., Giannini F., Hassinger S., and Ottcharova J., *Feature-based modelling by intergrating design and recognition approaches,* Computer Aided Design, Vol. 26, No. 8, pp646-653, 1994.

Descotte Y. and Latombe J., *GARI: A problem solver that plans how to machine mechanical parts,* IJCAI 7, Vancouver, pp766-772, 8/1981.

Dixon J.R., *Design with features: building manufacturing knowledge into more intelligent CAD systems,* Proceedings of the ASME Conference Manufacturing International '88, Atlanta, USA, 1988.

Doumeingts G., Vallespir B. and Chen D., *Methodologies for designing CIM systems: a survey,* Computers in Industry, Vol. 25, No. 3, pp263-280, 1995.

Duerr H., Wauer A., Warnecke H.J., and Muthsam H., *Knowledge based generative CAPP overcoming constraints of part families - an approach to much more flexibility in CAPP,* SME Technical Report Series, Dearborn, Michigan, USA, 1994.

Dunn M.S. and Mann W.S., *Computerised Production Process Planning,* Proceedings of the 15th Numerical Control Society Annual Meeting and Technical Conference, Chicago 4 / 1978.

El Maraghy H.A., Agerman E., Davies B.J. et al., *Evolution and future perspectives of CAPP,* CIRP Annals, Vol. 42, No. 2, pp739-751, 1993.

El-Midany T.T and Davies B.J., *AUTOCAP - A Dialogue System for Planning the Sequence of Operations for Turning Components,* Journal of Engineering and Applied Science, Vol. 1, No. 3, Pergamon Press, 1982.

Eskicioglu H. and Davies B.J., *An interactive process planning system for prismatic parts (ICAPP),* CIRP Annals, 32/1/1983.

Eversheim W., *Computer Aided Programming of NC Machine Tools by Using the System AUTAP - NC,* CIRP Annals, pp323-327, 30/1/1982.

Eversheim W. and Schneewind J., *Computer aided process planning - state of the art and future development,* Robotics and Computer Integrated Manufacturing, Vol. 10, No. 1-2, pp65-70, 1993.

Falcidieno B. and Giannini F., *Extraction and organisation of form features into a structured boundary model,* Proceedings of Eurographics, Amsterdam, pp249-259, 1987.

Fuh J.Y.H., Ji P., and Zhang Y.F., *Future development trends in CAM/CAPP-NC systems,* International Journal of Computer Applications in Technology, Vol. 8, No. 3-4, pp203-210, 1995.

Furth I.B., *Automated Process Planning,* NATO ASI Series, Vol F49, Computer Integrated Manufacturing, Springer-Verlag, pp37-64, 1988.

Gao J.X. and Case K., *A design by features approach to the building of feature data models for process planning,* Computer Aided Production Engineering (CAPE) 8, University of Edinburgh, 8/1992.

Gao J.X. and Huang X.X., *Product and manufacturing capability modelling in an integrated CAD / process planning environment,* International Journal of Advanced Manufacturing Technology, Vol. 11, No. 1, pp43-51, 1996.

Gindy N.N.Z., *A Hierarchical Structure for Form Features,* International Journal of Production Research, Vol. 27, No. 12, 1989.

Gindy N.N.Z. and Huang S.X., *Feature-Based Planning Data Model for Generative Planning Systems,* 29th International Matador Conference, pp37-44,

Manchester, 4/1992.

Gindy N.N.Z., Huang S.X. and Ratchev T.M., *Feature-Based Component Model for Computer Aided Process Planning Systems,* Symposium on Feature-based approaches to design and process planning, Loughborough, 9/1991.

Gindy N.N.Z., Huang S.X. and Ratchev T.M., *Feature-Based Component Model for Computer Aided Process Planning Systems,* International Journal of Computer Integrated Manufacturing, Vol. 6, Nos. 1&2, pp20-26, 1993.

Giusti F. and Santochi M., *KAPLAN: a Knowledge Based Approach to Process Planning of Rotational Parts,* CIRP Annals, Vol.38/1, pp481-484, 1989.

Groover M.P., *Automation, Production Systems and Computer Integrated Manufacturing,* Prentice-Hall International Inc., 1987.

Gu P., *Design of Cellular Manufacturing Systems: A Heuristic Approach,* Proceedings of the ASME International Conference on Computers in Engineering, pp177-184, NY, USA, 1991.

Gu P. and Norrie D.H., *Intelligent Manufacturing Planning,* Chapman & Hall, 1995.

Gu P. and Norrie D.H., *Intelligent Manufacturing Systems,* Chapman and Hall, 1996.

Gu P. and Zhang Y., *Operation sequencing in automated process planning,* Journal of Intelligent Manufacturing, Vol. 7, No. 1, pp17-28, 1993.

Gu P., Zhang Y. and Norrie D.H., *Object-oriented product modelling for Computer Integrated Manufacturing,* International Journal of Computer Integrated Manufacturing, Vol. 7, No. 1, pp17-28, 1994.

Gupta T., *An expert system approach in process planning: Current development and*

its future, Computers In Industry Engineering, Vol. 18, No. 1, pp69-80, 1990.

Ham I. and Lu C.Y., *Computer Aided Process Planning: The Present and the Future,* CIRP Annals, Vol. 37/2, pp591-601, 1988.

Henderson M.R., *Extraction of Feature Information from Three Dimensional CAD Data,* PhD Thesis, Purdue University, USA, 1984.

Henderson M.R., Chuang S.H., Ganu P., and Gavankar P., *Graph-based feature extraction,* Arizona State University, USA, 1990.

Herbert P.J., Hinde C.J., Bray A.D., Launders V.A., Round D. and Temple D., *Feature recognition within a truth maintained process planning system,* International Journal of Computer Integrated Manufacturing, Vol. 3, No. 2, 1990.

Houtzeel A., *CAPP perspective,* Modern Machine Shop, Vol. 68, No. 6, pp72-79, Nov 1995.

Huang S.H. and Zhang H., *Using neural networks for process planning,* Proceedings of the International Conference on Intelligent Manufacturing, Vol. 2620, pp429-440, Wuhan, China, 1995.

ISO 284, *Integrated Product Information Model - ISO STEP Baseline Requirements Document,* ISO Central Secretariat, Geneva, Switzerland, 10/1988.

Iwata K. and Fukuda Y., *KAPPS: Know - how and knowledge assisted production planning system in the machining shop,* 19th CIRP International Seminar on Manufacturing Systems, Penn. State, USA, 6/1987.

Iwata K. and Sugimur N., *Development of product model for design and manufacturing of machine products,* Proceedings of the 14th NAMRC, pp528-534, 1986.

Iwata K., Matsuo E., and Onosato M., *Development of new methodology for*

mechanical design by symbolised elements with movement, CIRP Annals, Vol. 41, No. 1, 1992.

Jared G.E.M., *Geometric Reasoning for Economic Manufacture,* Report on SERC Grant No GR/F/70785, 1991.

Jones P.F., *CAD/CAM: features, applications and management,* Mc Millan, 1992.

Joseph A.T. and Davies B.J., *EXCAP. An expert process planning system for turned components,* First International Conference on Expert Planning Systems, No. 322, pp130-135, Brighton, 1990.

Joshi S. and Chang T.C., *Graph-based heuristics for recognition of machined features from a 3D solid model,* Computer Aided Design, Vol. 20, No. 2, pp58-66, 1988.

Joshi S. and Chang T.C., *Feature extraction and feature-based design approaches in the development of design interface for process planning,* Journal of Intelligent Manufacturing, Vol. 1, pp1-15, 1990.

Jovanoski D. and Muthsam H., *Workpiece modelling for computer-aided process planning,* International Journal of Advanced Manufacturing Technology, Vol. 10, No. 6, pp404-410, 1995.

Kamrani A.K., Sferro P., and Handelman J., *Critical issues in design and evaluation of computer aided process planning systems,* Computers and Industrial Engineering, Vol. 29, No. 1-4, pp619-623, 1995.

Kalpakjian S., *Manufacturing Engineering and Technology,* Addison-Wesley, 1995.

Kiritsis D., *Review of knowledge-based expert systems for process planning. Methods and problems,* International Journal of Advanced Manufacturing Technology, Vol. 10, No. 4, pp240-262, 1995.

Kramer T.R., *Process planning from a milling machine from a feature based design,* Proceedings of the ASME Conference Manufacturing International '88, pp179-189, Atlanta, USA, 1988.

Krigman R., *Selecting the next generation solid modelling-based integrated mechanical CAE/CAD/CAM system,* AUTOFACT Conference Proceedings, Dearborn, Michigan, USA, 1992.

Krueger T.J., *How computer solid modelling is altering the process of detail design,* American Society of Mechanical Engineers, PD, Vol. 72, pp109-113, 1995.

Kusiak A., *The generalised Group Technology concept,* International Journal of Production Research, Vol. 25, No. 4, pp561-569, 1987.

Kusiak A., *Intelligent Manufacturing Systems,* Prentice-Hall, 1990.

Kyprianou L.K., *Shape classification in computer aided design,* PhD thesis, University of Cambridge, UK, 1980.

Laako T. and Mantyla M., *Feature modelling by incremental feature recognition,* Computer Aided Design, Vol. 25, No. 8, pp479-491, 1993.

Lee Y. and Fu K., *Machine understanding of CSG: Extraction and unification of manufacturing features,* IEEE Computer Graphics and Applications, Vol. 7, No. 1, pp20-32, 1987.

Lenau T. and Alting L., *XPLAN - an expert process planning system,* 2nd International Expert Systems Conference, London, UK, 10/1986.

Lenau T. and Mu L., *Feature based approaches within CAD and CAPP - a literature survey,* The Institute of Manufacturing Engineering, 1991.

Li R.K. and Yu M.H., *A framework for prismatic part data generation unit machine loop concept,* International Journal of Computer Integrated Manufacturing,

Vol. 3, No. 2, 1990.

Linardakis S., *Reference Listings of the BUCADIP and BUFIP Programs (version 9.0)*, Internal Report, School of Mechanical Engineering, Bath University, 1995.

Link C.H., *CAPP CAM-I automated process planning system*, Proc. of the 1976 NC Conference, CAM-I Inc, Arlington, Texas, USA, 1976.

Mantyla M., *An Introduction to Solid Modelling*, Computer Science Press, Maryland, 1988.

Mantyla M., Nau D., and Shah J., *Challenges in feature-based manufacturing research*, Communications of the ACM, Vol. 39, No. 2, pp77-85, USA, 1996.

Mason H., *STEP Part 1: Overview and Fundamental Principles*, ISO TC184/SC4/WG6 N14, 3/1991.

Meeran S. and Pratt M.J., *Automated feature recognition from 2D drawings*, Computer Aided Design, Vol. 25, pp7-17, 1993.

Meeran S., Pratt M.J. and Kay J.M., *The use of PROLOG in the automatic recognition of manufacturing features from 2D drawings*, Engineering Applications in Artificial Intelligence, Vol. 6, No. 5, pp409-423, 1993.

Mei J., Zhang H.C. and Oldham W.J.B., *Neural network approach for datum selection in computer-aided process planning*, Computers in Industry, Vol. 27, No. 1, pp53-64, Sep 1995.

Milacic V.R., *SAPT- Expert system for manufacturing process planning*, Computer Aided/Intelligent Process Planning, ASME WAM, Vol. 19, USA, 1985.

Mileham A.R., Isik I., Zhang Y.F., and Rustom E., *The Development of a Generative Process Planning System*, Advanced Manufacturing Seminar Series, Coventry Polytechnic, UK, 1988.

Mitchell F.H., *CIM systems: an introduction to Computer Integrated Manufacturing*, Prentice-Hall, 1991.

Mortensen K.S and Belnap B.K., *A Rule-Based Approach Employing Feature Recognition for Engineering Graphics Characterisation*, Computer Aided Engineering Journal, 12/1989.

Narayan S.V. and Ling Z.K., *Heuristics-based feature recognition: a graph approach*, ASME Design Engineering Division, Vol. 69, No. 1, pp299-306, 1994.

Nau D.S and Chang T.C., *A knowledge based approach to generative process planning*, Symposium of Computer Aided/Intelligent Process Planning, ASMI Winter Meeting, Miami, Florida, USA, 1985.

OD Engineering Systems, *SOFIE2*, Coventry, United Kingdom

PAFEC, *LOCAM*, Nottingham, United Kingdom.

Pande S.S. and Prabhu B.S., *An expert system for automatic extraction of machining features and tooling selection automata*, Computer Aided Engineering Journal, Vol. 7, No. 4, pp99-103, 1990.

Pande S.S. and Walkevar M.C., *PC-CAPP - a computer assisted process planning system for prismatic components*, Computer Aided Engineering Journal, pp133-138, 8/1989.

Pande S.S. and Walkevar M.C., *PRICAPP: A computer assisted process planning system for prismatic components*, International Journal of Production Research, Vol. 28, No. 2, pp279-292, 1990.

Parkinson A., *The use of solid models in BUILD as a database for NC machining*, Proceedings of Prolamat, Paris, pp293-299, 1985.

Parry-Barwick S. and Bowyer A., *Feature Technology*, Technical Report

001/1993, School of Mechanical Engineering, University of Bath, UK, 1993.

Perng D.B., Chen Z., and Li R.K., *Automatic 3D machining feature extraction from 3D CSG solid input,* Computer Aided Design, Vol. 22, No. 5, pp285-295, 1990.

Pratt M.J., *Applications of feature recognition in the product life cycle,* International Journal of Computer Integrated Manufacture, Vol. 6, Nos. 1,2, pp13-19, 1993.

Pratt M.J. and Wilson P.R., *Requirements for support of form features in a solid modelling system,* CAM-I, R-85-ASPP-01, 1985.

Puttre M., *Computer aided manufacturing. Sculpting parts from stored patterns,* Mechanical Engineering, Vol. 114, No. 4, pp66-70, Apr 1992.

Rahman W.A., Harun W., and Case K., *Object-Oriented Feature-Based Design,* Proceedings of the Tenth National Conference on Manufacturing Research, pp294-298, Loughborough, 1994.

Rao H.A. and Gu P., *Design of cellular manufacturing systems: a neural net approach,* International Journal of Systems Automation: Research and Applications, Vol. 2, pp407-427, 1992.

Rembold U., Nuaji B.O., and Storr A., *Computer Integrated Manufacturing and Engineering,* Addison-Wesley, 1993.

Requicha A.A.G. and Chen S.C., *Representation of geometric features, tolerances and attributes in solid modellers based on constructive geometry,* IEEE Journal of Robotics and Automation, Vol. RA-2, No. 3, pp156-166, 1986.

Requicha A.A.G. and Vandenbrande J., *Automated Systems for Process Planning and Part Programming,* IFS Publications Ltd., Springer-Verlag, 1988.

Requicha A.A.G. and Vandenbrande J., *Form features for mechanical design and*

manufacturing, ASME Computers In Engineering Conference, pp47-52, Anaheim, USA, 8/1989.

Roy U., Bharadwaj B., Chavan A. and Mohan C.K., *Development of a feature based expert manufacturing process planner,* Proceedings of the 7th International Conference on Tools with Artificial Intelligence, pp63-70, IEEE, Herndon, VA, USA, 1995.

Rosenfield H., Rodriguez S.R., and De Oliveira J.F.G., *AI Based CAPP Methods Integrated Into a CAPP Environment,* Proceedings of the 29th International Matador Conference, pp53-56, 1992.

Rustom E.A., *BEPPS-GSCAPP Generative System of Computer Aided Process Planning for Prismatic Components,* PhD Thesis, Bath University, 1992.

Rustom E.A. and Mileham A.R., *The Development of a Generative System of Computer Aided Process Planning for Prismatic Components,* Proceedings of the 5th National Conference on Production Research, pp259-263, Huddersfield, UK, 1989.

Rustom E.A. and Mileham A.R., *Feature Ordering and Operation Sequencing for Automated Process Planning,* Proceedings of the 6th National Conference on Production Research, pp259-263, Glasgow, UK, 1990.

Rustom E.A. and Mileham A.R., *Automated Decision Making for Process Planning Systems,* Proceedings of the Joint International Conference FAIM 92, Flexible Automation and Information Management, Virginia, USA, 1992.

Sabourin L. and Villeneuve F., *OMEGA, an expert CAPP system,* Computers in Engineering, Proceedings of the International Conference and Exhibit, Vol. 1, pp261-269, ASME, New York, USA 1994.

Sabourin L. and Villeneuve F., *OMEGA, an expert CAPP system,* Advances in Engineering Software, Vol. 25, No. 1, pp51-59, Jan 1996.

Sack J.C.F., *CAM - I's experimental planning system XPS - 1,* Autofact 5 Conference Proc, Detroit, Michigan, USA, 11/1983.

Sakal R.L. and Chow J.G., *Integrated intelligent process planning system for prismatic parts using PC-based CAD and CAM software packages,* International Journal of Advanced Manufacturing Technology, Vol. 9, No. 3, pp166-174, 1994.

Sakurai H. and Gossard D.C., *Shape feature recognition from 3D solid models,* ASME International Computers in Engineering Conference, San Francisco, USA, 1988.

Schaffer G., *GT via Automated Process Planning,* American Machinist, pp119-122, 5/1980.

Schildt H., *Using Turbo C++,* McGraw-Hill, 1990.

Shah J.J., *Assesment of features technology,* Computer Aided Design, Vol. 23, No. 5, pp331-343, 1991.

Shah J.J. and Mathew A., *Experimental Investigation of the STEP Form-Feature Information Model,* Computer Aided Design, Vol. 23, No.4, 5/1991.

Shah J.J. and Rogers M.T., *Feature based modelling shell: design and implementation,* Computers In Engineering, 1/1990.

Shah J., Sreevalsan P. and Mathew A., *Survey of CAD / feature-based process planning and NC programming,* Computer-Aided Engineering Journal, Vol. 8, No. 1, pp25-33, Feb. 1991.

Shah J.J., Sreevalsan P., Rogers M., Billo R. and Mathew A., *Current Status of Features Technology,* CAM-I Report R88-GM-04, Arlington, USA, 1988.

Shen Y. and Shah J.J., *Feature recognition by volume decomposition using half-space partitioning,* ASME Design Engineering Division, Vol. 69, No. 1, pp575-583,

1994.

Singh N., *Systems approach to computer integrated design and manufacturing,* Wiley NY, 1996.

Sivayoganathan K., Balendran V., Czerwinski A., Keats J., Leibar A.M. and Seilar A. *CAD/CAM Data Exchange Application*, Proceedings of the Ninth National Conference for Manufacturing Research, pp306-310, 1993.

Sormaz D.N. and Khoshnevis B., *Knowledge representation for automated process planning,* Proceedings of the IEEE International Symposium on Assembly and Task Planning, pp34-39, Pittsburgh, PA, USA, 1995.

Staley S.M., Henderson M.R., and Anderson D.C., *Using syntactic pattern recognition to extract feature information from a solid modelling database,* Computers in Mechanical Engineering, Vol. 2, No. 2, pp61-65, 1983.

Subrahmanyam S. and Wozny M., *Overview of automatic feature recognition techniques for computer aided process planning,* Computers in Industry, Vol. 26, No. 1, pp1-21, 1995.

Suh N.P., *The Principles of Design,* Oxford University Press, 1990.

Taiber J.G., *Development of an optimization method for determination of process sequences considering prismatic workpieces,* Computers in Engineering, Proceedings of the International Conference and Exhibit, Vol. 1, pp271-280, ASME, New York, USA 1994.

Tang K. and Woo T., *Algorithm aspects of alternating sum of volumes. Part 1: Data structure and difference operation,* Computer Aided Design, Vol. 23, No. 5, pp357-366, 1991.

Tang K. and Woo T., *Algorithm aspects of alternating sum of volumes. Part 2: Nonconvergence and its remedy,* Computer Aided Design, Vol. 23, No. 6, pp435-

443, 1991.

Trego L., *Computer aided manufacturing software,* Automotive Engineering, Vol. 103, No. 11, pp73-81, 1995.

Tulkoff J., *Process planning in the computer age,* Machine and Tool Blue Book, 11/1981.

Turner G.P. and Anderson D.C., *An object-oriented approach to interactive feature-based design for quick turnaround manufacturing,* Proceedings of ASME Computers in Engineering Conference, Vol. 1, San Fransisco, USA, 1988.

Udo G.J. and Udoka S.J., *Network management: a critical success factor for implementing computer integrated manufacturing,* Computers and Industrial Engineering, Vol. 23, No. 1-4, pp409-412, 1992.

Vandenbrande J.H. and Requicha A.A., *Spatial reasoning for automatic recognition of interacting form features,* Proceedings of the ASME Computers in Engineering Conference, pp251-256, NY, USA, 1990.

Van Houten F.J.A.M., Van't Erve A.H. and Kals H.J.J., *PART, a feature based CAPP system,* University of Twente, SPIN-project PART, Netherlands, 2/1989.

Van't Erve A.H. and Kals H.J.J., *XPLANE: a generative Computer Aided Process Planning System for part manufacturing,* CIRP Annals, pp325-329, 35/1/1986.

Vaquero H.R.M.S., *Information integration in computer integrated manufacturing (CIM),* Journal of the Brazilian Society of Mechanical Sciences, Vol. 17, No. 1, pp35-48, 1995.

Vogel S.A and Adlard E.J., *The AUTOPLAN Process Planning System,* Proceedings of the 18th Numerical Control Society Annual Meeting and Technical Conference , Dallas, Texas, USA, pp422-429, 5/1981.

Vosniakos G.C. and Davies B.J., *An IGES Post-Processor for Interfacing CAD and CAPP of 2-1/2D Prismatic Parts* International Journal of Advanced Manufacturing Technology, pp. 135-164, 5/1990.

Wang E., *Using automatic feature recognition to interface CAD to CAPP,* Proceedings of the International Computers in Engineering Conference and Exhibit, Vol. 1, pp215-231, 1992.

Wang H.P. and Lin C.A., *Automated Generation of NC Part Programs for Turned Parts Based on 2D Drawing Files,* International Journal of Advanced Manufacturing Technology, 2(3), pp23-35, 1987.

Wang H.P. and Wysk R.A., *Intelligent Reasoning for Process Planning,* Computers In Industry 8, pp293-309, 1987.

Weston F.C., *Three dimensions of CIM,* Production and Inventory Management Journal, Vol. 35, No. 1, pp59-61, 1995.

Wierda L.S., *Linking design, process planning and cost information by feature-based modelling,* Journal of Engineering Design, Vol. 2, No. 1, pp3-9, 1991.

Woodwark J.R., *Computing Shape,* Butterworths, 1986.

Wysk R.A., *An Automated Process Planing And Selection Program: APPAS,* PhD Thesis, Purdue University, West Lafayette, Indiana, USA, 1977.

Yuen M.F., Tan S.T., Sze W.S., and Wong W.Y., *An octree approach to rough machining,* Proceedings of the Institute of Mechanical Engineers, (B3) 201, pp157-163, 1987.

Yura K., Ohashi K., Nakajima M., Yoshimura M., Ota M., and Hitomi K., *Strategic planning for CIM to enhance the competitive ability,* Computers and Industrial Engineering, Vol. 27, No. 1-4, pp127-130, 1994.

Zhang Y., *The Automation of Process Planning from the CAD Product Model to NC Programs for Rotational Parts,* PhD Thesis, Bath University, UK, 1991.

Zhang Y., *Integrated process planning system in CIMS environment,* China Mechanical Engineering, Vol. 4, No. 4, pp24-26, Aug 1993.

Zhang Y. and Mileham A.R., *BEPPS-NC: A Rule Based Expert System for Process Planning,* Proceedings of the 5th National Conference on Production Research, pp333-357, UK, 1989.

Zhang Y. and Mileham A.R., *A CAD Interpreter for interfacing CAD and CAPP of 2-D Rotational Parts,* Proceedings of the 6th International Conference on CAD/CAM, Robotics and Factories of the Future, UK, 1991.

Zhang Y. and Mileham A.R., *An Automated Generator of NC Part Programs for Rotational Parts from CAD Product Model,* Proceedings of the Seventh National Conference on Production Research, pp11-15, 1991.

Zhang Y.F., Nee A.Y.C. and Fuh J.Y.H., *Hybrid approach to computer aided process planning for prismatic parts,* Computers in Engineering, Proceedings of the International Conference and Exhibit, Vol. 1, pp437-442, ASME, New York, USA 1994.

Zhao Z., *Process planning with multi-level fuzzy decision-making,* Computer Integrated Manufacturing Systems, Vol. 8, No. 4, pp245-254, Nov 1995.

Zhao Z. and Baines R.W., *Conjugative coding design and manufacturing information for Computer Integrated Manufacturing,* International Journal of Production Research, Vol. 30, No. 10, pp2313-2333, 1992.

Zhao Z. and Baines R.W., *Conjugative Inference Mechanism in Generative Process Planning,* Proceedings of the Eighth National Conference on Manufacturing Research, Birmingham, UK, pp253-258, 1992.

Zhao Z. and Baines R.W., *CCSPLAN: A Generative Process Planning System,* Proceedings of the Thirtieth International Matador Conference, Manchester, pp527-534, 1993.

Zhao Z. and Baines R.W., *Computer Aided Process Planning with Interface to 3D Wireframe Modelling System,* Proceedings of the Tenth Conference of the Irish Manufacturing Committee IMC 10, University College Galway, Ireland, pp669-680, 1993.

Zhao Z., Blount G.N., Jones R.M. and Baines R.W., *Representation of Surfaces in 3-D Wireframe Models,* Proceedings of the Thirtieth International Matador Conference, Manchester, pp573-579, 1993.

Zust R. and Taiber J., *Knowledge Based Process Planning System for Prismatic Workpieces in a CAD/CAM Environment,* CIRP Annals, 39/1/1990.

APPENDIX A
SETTING UP THE AUTOCAD DRAWING ENVIRONMENT FOR EFFECTIVE FEATURE IDENTIFICATION: A STEP BY STEP GUIDE

Note: the following applies to AUTOCAD version 10.2, used throughout the research work. Other versions or packages might have different commands. *Click* refers to clicking with the mouse pointer. The keyboard can be used directly to provide input to the *Command:* prompt. *Ctrl+C* cancels the current command if a mistake is made.

A1. Setting up the Basic Environment

Having started AUTOCAD and selected option 1 from the Main Menu (Begin a NEW drawing) the name of the drawing must be typed in. Once this has been accomplished, the drawing editor environment is activated. From here:

1. Click on Setup.
2. Click on metric (for metric units).
3. Click on Full (for full scale drawing).
4. Click on 297x210 (for the drawing paper size; this can be up to A2 if required).
5. On the *Command:* prompt at the bottom of the screen type *erase*.
6. Select the white border that just formed around the drawing border and press the *Spacebar* to erase it.
7. From the *Settings* menu at the top of the screen choose *Drawing Aids...*
8. From the dialogue box click and type a 10 spacing for X and Y Snap and Grid and check the Snap and Grid checkboxes.
9. Click on OK to activate the settings.

A2. Setting up the Layered Views and Linetypes

1. From the *Settings* menu choose *Modify Layer...*
2. From the dialogue box click on New Layer.
3. Type PLAN and click on OK (this creates the PLAN layer; layer 0, the front one, is already there).
4. Repeat steps 2 and 3 but now type END instead (for the end layer).

The layers are now set up. Click on the Current checkbox and then on OK to select the layer to draw on. Now to select the appropriate line type, which must first be loaded:

1. On the *Command:* prompt type *linetype*.
2. Type *load*.
3. Type *hidden* or *dashed* (BUFIP accepts both types).
4. Press *Enter* at the *File to search:* prompt. Press *Ctrl+C*.
5. From the *Settings* menu choose *Entity Creation...*
6. From the dialogue box click on linetype, and then check the desired line type checkbox (CONTINUOUS, DASHED or HIDDEN).
7. Click on OK and then OK again.

The component drawing can now be drawn. It is reminded that care should be taken on which is the current active drawing layer (shown at the top left hand corner of the screen) and which line type is currently being used (checked using the *Settings* menu and then *Entity Creation...*).

A3. Setting up the Environment for Dimensioning and Tolerancing

First let's set up the tolerancing layers:

1. From the *Settings* menu choose *Modify Layer...*
2. From the dialogue box click on New Layer.
3. Type DIMF and click on OK (this creates the DIMensional Front layer).
4. Repeat steps 2 and 3 twice but now type DIMP and DIME. This creates the other

two dimensional layers.

Now to set up the dimensioning variables:

1. From the *Command:* prompt type *setvar*.
2. Type *dimaso* and then enter 0 when asked for the new value.
3. Press the *Spacebar* to repeat the *setvar* command and now type *dimtol*. Enter 1 when asked for the new value.
4. Press the *Spacebar* and now type *dimtxt*. Enter 3 when asked for the new value.
5. Press the *Spacebar* and now type *dimasz* and enter 3 again.
6. Press the *Spacebar* and type *dimtm*. This sets the lower limit tolerance value, so type in the desired value for that.
7. Press the *Spacebar* and type *dimtp*. This sets the upper limit tolerance value, so type in the desired value for this.

The tolerancing system is now set. Dimensioning and tolerancing can be activated using the *Dim* command at the *Command:* prompt, then typing *hor* (horizontal) or *ver* (vertical) and following the prompts for horizontal or vertical dimensioning. Again, care should be taken on which is the current active dimensional layer, and which tolerance values are being used.

A4. Producing a DXF Output File

1. From the *Command:* prompt type *dxfout*.
2. Enter the desired file name (without any extensions).
3. Press *Enter* when prompted for decimal places of accuracy. This produces 6 decimal places of accuracy. The DXF file is then created.

APPENDIX B

PUBLISHED PAPERS FROM THE WORK

1. *A CAD Interpreter for Prismatic Components*, Proceedings of the Eighth National Conference on Manufacturing Research, pp21-26, Birmingham, UK, September 1992.

2. *Manufacturing Feature Identification for Prismatic Components from CAD DXF Files*, Proceedings of the Ninth National Conference on Manufacturing Research, pp37-42, Bath, UK, September 1993.

3. *A Strategy for Extracting Manufacturing Features for Prismatic Components from CAD*, Proceedings of the Tenth National Conference on Manufacturing Research, pp299-304, Loughborough, UK, September 1994.

4. *A Methodology for Identifying Features for Prismatic Components from CAD Files*, Proceedings of the Eleventh National Conference on Manufacturing Research, pp159-164, Leicester, UK, September 1995.

5. *A Methodology for Extracting Manufacturing Features for Prismatic Components from CAD DXF Files*, Proceedings of the Twelfth Conference of the Irish Manufacturing Committee IMC 12 on Competitive Manufacturing, pp71-78, Cork, Ireland, September 1995.

A CAD Interpreter for Prismatic Components

S. Linardakis and A.R. Mileham

School of Mechanical Engineering, University of Bath, Claverton Down, BATH BA2 7AY

Abstract

A novel approach for interfacing CAD and CAPP for prismatic components is discussed. The CAD interpreter, under development, uses the industry standard DXF (Drawing Interchange File) format to extract drawing entities (eg lines, circles, arcs) and their geometric information. It then classifies and formalises them into machining features. The output product model file can be used by a computer aided process planning system directly. A typical example using the interpreter is presented.

1. Introduction

In manufacturing systems, the task of process planning is to determine the sequence and procedure of the individual manufacturing processes required to transform raw material into a finished part. Process planning, which is the link between design and manufacturing (in recent years Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM)) is probably the most difficult function in the area of production planning. In order to achieve integration of CAD and CAM, and hence increase productivity in manufacturing, this creative task must ideally be performed automatically. One of the important challenges towards achieving this goal is interfacing the CAD database with the Computer Aided Process Planning (CAPP) system. This would enable automatic product definition data input to the planning module. A considerable number of research approaches have been developed in this area (Alting L. and Zhang H.C. (1989)).

An IGES (Initial Graphics Exchange Specification) file format post processor has been reported (Vosniakos G.C. and Davies B.J. (1990)), which can extract the data that defines a 2-1/2D prismatic component, based on a wire-frame IGES file. However, the awkward points of the IGES standard makes it impossible to transfer the full context of a product model, unless a series of restrictions are applied. An identifier able to extract geometric information from a part data file (generated using a 2D wire-frame CAD modeller) has also been developed (Wang H.P. and Wysk R.A. (1987)). However, information such as tolerance, surface roughness etc is still typed interactively. Zhang (Zhang Y. and Mileham A.R. (1990), Zhang Y. (1991)) developed an interpreter using the DXF (Drawing Interchange File) format for 2D rotational parts that can extract tolerance information automatically. Although all these approaches have contributed

significantly to this domain, research on the topic is far from being conclusive.

The CAD interpreter described in this paper, is being developed to identify the geometric and tolerance data that define a prismatic component from a DXF file format. It aims at minimising human intervention on the interface of CAD and CAPP.

2. The DXF File

In a CAD system, the drawing database is stored in a system specific, very compact format. However, in order to enable data exchange between different CAD packages a "Drawing Interchange File" format has been defined (Autodesk Inc.(Anon.) (1986)). Data files using this format are termed DXF files and are encoded in ASCII characters and numbers, so that they can be easily read directly by other programs. Most wire-frame CAD packages on the market have the ability to produce DXF files, thus making the DXF format an emerging industry standard.

A DXF file consists of four main sections :
(1) The HEADER section containing general information about the drawing.
(2) The TABLES section containing definitions of named items such as fonts, line types and layers.
(3) The BLOCKS section containing Block Definition entries (blocks are groups of drawing entities, for example lines, circles, arcs).
(4) The ENTITIES section containing information of all the drawing entities, such as geometric entities (lines, circles etc) and text (dimensions, tolerances etc), including any Block references.

The ENTITIES section is the most important part of the DXF file for an interpreter program, since it contains the data defining a component. Each entity type is uniquely specified by an entity type name. Its characteristics such as coordinates, dimensions etc are defined by parameters following its type name.

3. The CAD Interpreter

3.1 Methodology

From the process planning point of view, a component should be defined in terms of its manufacturing features (eg flat surfaces, pockets, holes) rather than pure geometric entities (lines, circles etc). The objective of the CAD interpreter under development is to convert the CAD geometric entity information into a manufacturing feature product model, which can then drive a process planning system directly.

There are two major tasks involved in this process (Chang T.C. (1990)): model decomposition and feature recognition. The decomposition task separates the features from a component model, while the recognition task identifies and classifies the characteristics of each feature. However, only geometric information can be translated using the above method.

In order to transform a CAD model into a manufacturing one, the interpreter proceeds in the following manner :

(1) The DXF file is read and only the necessary information extracted. The entity parameters are classified and stored into a data file for further processing.
(2) The entities which form the external profile of the component are identified and separated. The internal profile entities are then recognised (if there are any). Manufacturing features are deduced and classified.
(3) Tolerances are allocated to their relevant dimensions. Dimensioning text entries are used for tolerance allocation and error checking of the interpretation process.
(4) Processed information is stored in a format by which a manufacturing product model is defined.

A flow diagram of these operations can be seen in **Figure 1**. Each operation is discussed in further detail below.

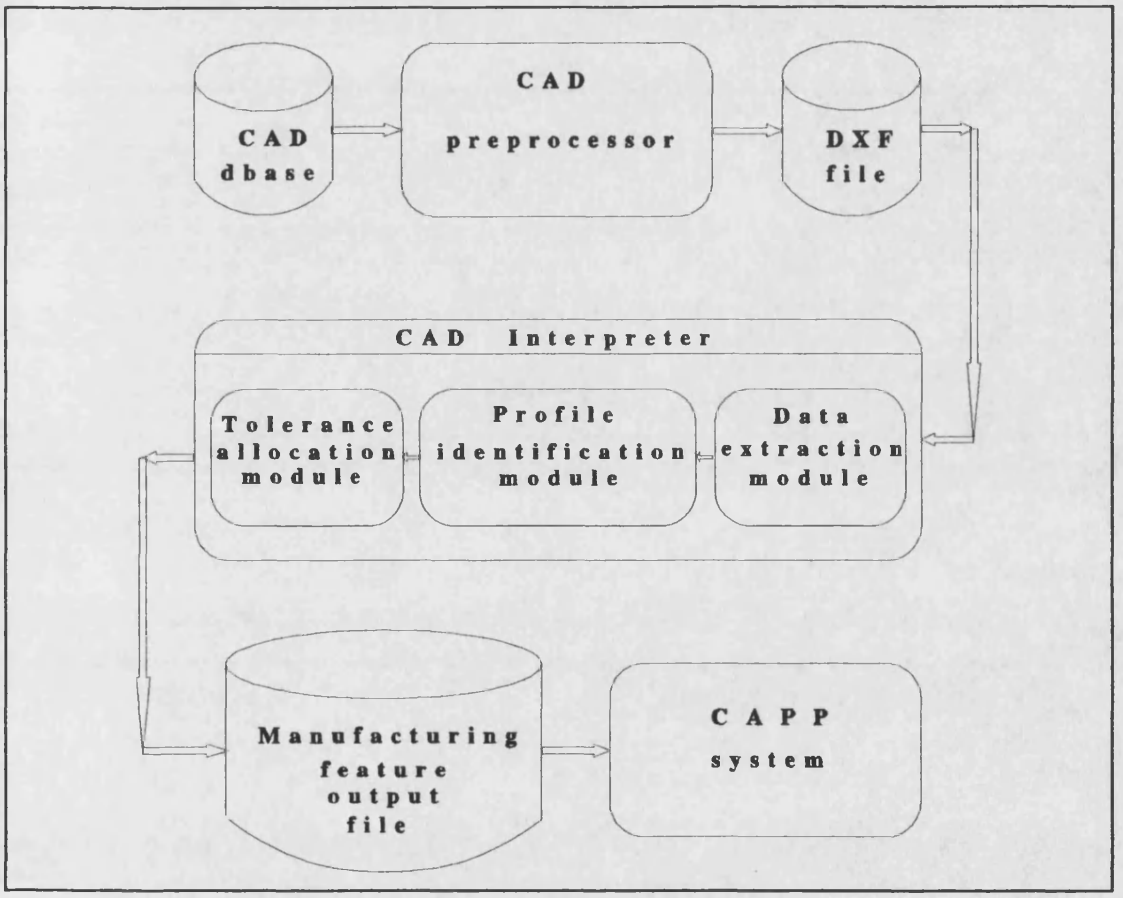


Figure 1 Flow Diagram of the CAD Interpreter

3.2 Data Extraction and Classification

As mentioned previously, the ENTITIES section of the DXF file is used by the interpreter to obtain component data information. An algorithm is being developed with the intelligence to extract each geometric entity together with its relevant characteristics. Thus, if an entity is, for example, identified as a line, its starting point and ending point

coordinates are then read, if it is a circle its centre point coordinates and radius are extracted, and so on. The entities, along with their characteristics, are next classified into groups of the same type (eg lines, circles) with each entity being allocated a number for future reference. The results are stored in an ASCII data file for further processing.

3.3 Profile Recognition and Manufacturing Feature Identification

In this stage, both the internal and external profile of the component is identified. For a prismatic part, complete manufacturing information can be provided by a three-view drawing in 2D space using orthographic projection. However, the duplication of the same features on different views of the drawing could confuse the interpretation process. This can be avoided by placing the different views of the drawing in different *layers*. Hence the front view can be in one layer, the plan in another, dimensioning lines and text in a third and so on. Since the layer that an entity belongs to is also reflected in its entry in the DXF file, the profile recognition and feature identification task becomes easier.

An algorithm is under development which identifies the profiles (external and internal) and manufacturing features of a component by correlating entities from its different views. The views are layered in a predefined convention (there is no current standard for layering views of a drawing). The external profile of the component is identified first, followed by the internal profile, if there is any. Manufacturing features are next deduced and classified.

3.4 Tolerance Allocation

Two types of tolerance appear in engineering drawings : dimensional and geometrical. The interpreter deals with dimensional tolerances only. A dimensional tolerance appears as a text entity in the ENTITIES section of the DXF file. Its basic size, along with witness lines information (tolerance string position, witness gap etc) also appear in the same entry. It is thus possible for the interpreter to allocate the tolerance to a feature. Dimensioning text entities in the DXF file are also used for verification of the entity data extraction process.

3.5 Output Format

The final results of the interpretation process are stored in an ASCII data file in which each machining feature occupies an entry. The entry contains the feature type and number, along with its dimensional parameters and tolerance. The data file can be easily accessed by a Computer Aided Process Planning system for further processing or modification.

4. CAD Interpreter Application Example

The interpreter is being developed on an IBM-486 compatible computer. The program is written in C and can thus be easily ported on a different computer operating system should the need arise. AUTOCAD (Autodesk Inc.(Anon.) (1986)) and DAXCAD (Practical Technology (Anon.) (1985)) are the CAD packages used to design components and produce the DXF files.

5. Conclusions

A CAD interpreter for prismatic components is under development. The interpreter uses the CAD industry standard DXF file format to automatically extract drawing entities and relevant information. It then processes and formulates the data into a manufacturing feature based product model output file, which can be directly accessed by a Computer Aided Process Planning system. The system is being validated over a range of prismatic components and represents a further step towards CAD/CAPP integration, opening a new research direction in the field of interfacing CAD and CAPP.

References

Practical Technology (Anon.) *DAXCAD (Rev 2.136), (c) 1985, 86, 87, 88.*

Autodesk Inc. (Anon.) *AUTOCAD User Reference* 1986.

Wang H.P. and Wysk R.A. *Intelligent Reasoning for Process Planning* Computer In Industry 8 1987, pp. 293-309.

Alting L. and Zhang H.C. *Computer Aided Process Planning- The State of the Art Survey,* International Journal of Production Research 1989 Vol. 27 No. 4, pp. 553-585.

Chang T.C. *Expert Process Planning for Manufacturing* Addison-Wesley Publishing Company 1990.

Vosniakos G.C. and Davies B.J. *An IGES Post-Processor for Interfacing CAD and CAPP of 2-1/2D Prismatic Parts* International Journal of Advanced Manufacturing Technology 5 1990, pp. 135-164.

Zhang Y. and Mileham A.R. *A CAD Interpreter for Interfacing CAD and CAPP of 2D Rotational Parts* Proceedings of the Sixth International Conference on CAD/CAM, Robotics etc 1990, pp.333-336.

Zhang Y. *The Automation of Process Planning from the CAD Product Model to NC Programs for Rotational Parts* PhD Thesis, Bath University 1991.

Manufacturing Feature Identification for Prismatic Components from CAD DXF Files

S. Linardakis and A.R. Mileham

School of Mechanical Engineering, University of Bath, Claverton Down, BATH BA2 7AY

Abstract

A novel approach for interfacing Computer Aided Design (CAD) and Computer Aided Process Planning (CAPP) for prismatic components is discussed. The analysis presented forms part of a CAD interpreter under development, which uses the industry standard DXF (Drawing Interchange File) file format to translate CAD drawings into a manufacturing feature based output file, to be directly used by a CAPP system. Algorithms for identifying the external profile of a prismatic component and allocating surface roughness values to surfaces requiring finishing, are discussed.

1. Introduction

Two of the major activities within the development phase of industrial products is product design and planning of production. These two activities are often carried out in isolation within companies. Today's competitive situation requires that product development times are shortened, product quality improved, costs reduced, and environmental consequences minimised. To realise these goals, integration of the design and planning activities is considered essential.

Describing components by the use of features is seen by many as the key to genuine integration of the many aspects of design and planning of manufacture, particularly in a modern computer-controlled environment incorporating Computer Aided Design (CAD) and Computer Aided Process Planning (CAPP) (Brimson and Downey 1986). Features originate in the reasoning processes used in various design, analysis, and manufacturing activities (Cunningham and Dixon 1988) and are frequently strongly associated with particular application domains. Hence there are many different definitions of features. A broad definition in the engineering domain is given by (Pratt 1988) as: "A feature is a region of interest on the surface of a part".

The use of features on the design side ("design features") could relate to the fulfilment of functional requirements, the building of a geometric model, or as preparation for design analysis activities such as finite element analysis. On the planning side, activities such as process planning, assembly planning, manufacturing operations planning and part programming for numerically controlled machines could potentially be based upon

a feature representation of a component. In this case, features may be viewed differently by process planners or NC programmers as "manufacturing features", for example, a fixing hole may be considered as a drilled or bored hole etc (Shah and Rogers 1988).

From the design point of view, one of the attractions of geometric modelling within CAD is that a well constructed modeller is capable of representing all the geometric aspects of a component within a chosen domain. If a complete geometric description is available, then it is clearly possible to use computer methods to interrogate this information and transform it into any desired form. Thus collections of surfaces could be recognised as features and transformed into manufacturing features for CAPP purposes. Much research work has been done in this Feature Identification field.

2. Feature Identification from Wireframe CAD Data

Mortensen and Belnap (1989), describe a methodology employing feature recognition for rotational parts from a 2D CAD system database (although an interactive graphics system was used to generate the test data). However this system is limited as it lacked automatic dimension extraction. An IGES (Initial Graphics Exchange Specification) file format post processor has been reported (Vosniakos and Davies 1990), which can extract the data that defines a 2-1/2D prismatic component, based on a wire-frame IGES file. However, the awkward points of the IGES standard makes it impossible to transfer the full context of a product model, unless a series of restrictions are applied. An identifier able to extract geometric information from a rotational part data file (generated using a 2D wire-frame CAD modeller) has also been developed (Wang and Wysk 1987, 1988). However, information such as tolerance, surface roughness etc is still typed interactively. Zhang and Mileham (1990) developed an interpreter using the DXF format for 2D rotational parts that can extract tolerance information automatically. Although all these approaches have contributed significantly to this domain, research on the topic is far from being conclusive.

The work presented in this paper is part of ongoing research on a CAD Interpreter for Prismatic Components being developed at Bath University (Linardakis and Mileham 1992) aiming at minimising human intervention on the interface of CAD and CAPP. The interpreter identifies the geometric and tolerance data that define a prismatic component from a wireframe-based CAD DXF file. The DXF file format is an emerging industry standard, enabling data exchange between different CAD packages, hence by using it as an input, the interpreter retains compatibility. An algorithm has already been developed to extract and classify the geometric entities (lines, circles, arcs etc) of a drawing. The objective of the research work is to then convert the CAD geometric entity information into a manufacturing feature based model, which can be used to drive a process planning system directly.

3. External Profile Identification

For a prismatic part, complete manufacturing information can be provided by a three-view drawing in 2D space using orthographic projection. However, the duplication of the same features on different views of the drawing could confuse the interpretation process. This can be avoided by placing the different views of the drawing in different

layers. The views are layered in a predefined convention (there is no current standard for layering views of a drawing) as follows:

Layer 0	FRONT view
Layer 1	PLAN view
Layer 2	END view
Layer 3	DIMension and tolerancing layer
Layer 4	SURface roughness layer

Since the layer that an entity belongs to is also reflected in its entry in the DXF file, the profile recognition and feature identification task becomes easier.

An algorithm is under development which identifies the profiles (external and internal) of a component by correlating its entities from different views. The external profile of the component is identified first. This enables the interpreter to determine the shape envelope dimensions, and hence deduce the minimum shape envelope of the required raw material for the component to be machined.

In order to identify the external profile of the component, the interpreter proceeds in the following manner:

(1) The extracted line entities and their coordinates from the PLAN view are first examined. The X and Y edges of the component are formed by the lines with the minimum and maximum X and Y coordinates:

Line (Xmin, Ymin) to (Xmin, Ymax) forms an edge.
Line (Xmin, Ymax) to (Xmax, Ymax) forms an edge.
Line (Xmax, Ymax) to (Xmax, Ymin) forms an edge.
Line (Xmax, Ymin) to (Xmin, Ymin) forms an edge.

(2) The same method is applied to the extracted line entities and their coordinates from the FRONT view (Y coordinates being transformed to Z coordinates). Hence the Z dimension of the component is determined.

A graphical representation of this methodology can be seen in **Figure 1**. In the case of a feature being encountered at an edge (eg side pocket, slot etc) the interpreter compensates by extrapolating edge line coordinates to their corresponding minima and maxima, thus always forming a block shape envelope from which the required raw material shape envelope can be determined.

4. Surface Roughness Allocation

Apart from dimensioning and tolerancing information, surface roughness values are often required for determining process routes. Since AUTOCAD, the CAD modeller used for generating sample DXF files (Autodesk Inc. (Anon.) 1986) does not readily support

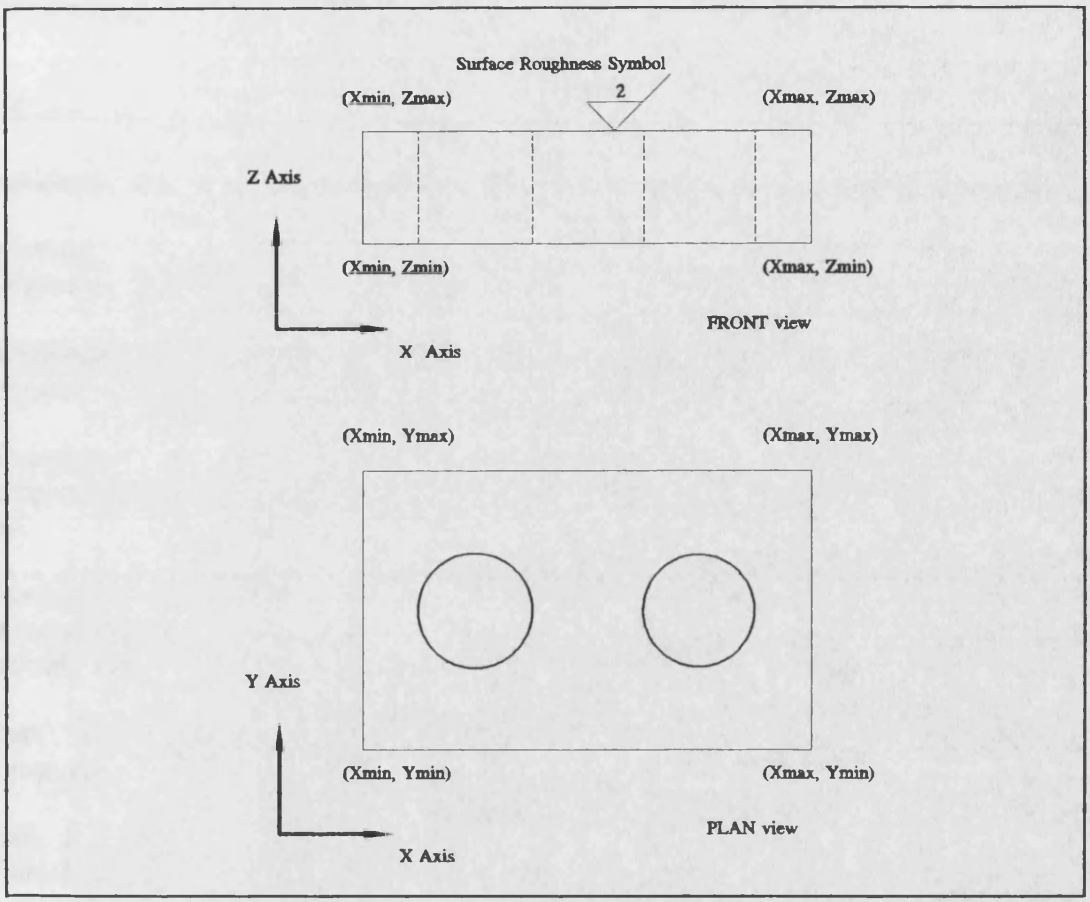


Figure 1 External profile identification methodology

surface roughness, an algorithm is being developed to address this problem.

This prerequisites the creation of a surface roughness layer on the component drawing (as mentioned previously). Provided the standard BS308-1972 surface roughness symbol is used (**Figure 1**) adjacent to the surface feature to be machined, it should be possible for the interpreter to extract this symbol and allocate it appropriately. In the case of component edges requiring finishing, an allowance on the shape envelope dimensions could be added to the length of the respective surface for machining operations. This methodology could be applied to tolerance allocation as well.

5. Conclusions

A CAD interpreter for prismatic components is under development. The interpreter uses the CAD industry standard DXF file format to automatically extract drawing entities and relevant information. It then proceeds to identify and determine the component shape envelope dimensions, and hence deduce the minimum shape envelope of the required raw material. An algorithm is also being developed for the automatic allocation of surface roughness values. The system is being validated over a range of prismatic components and

represents a further step towards CAD/CAPP integration.

References

Autodesk Inc. (Anon.) 1986. *AUTOCAD User Reference.*

Brimson J.A. and Downey P.J. 1986. *Feature Technology: A Key to Manufacturing Integration* CIM Review, Spring.

Cunningham J.J and Dixon J.R. 1988. *Designing with Features: The Origin of Features* Computers in Engineering, Vol 1.

Linardakis S. and Mileham A.R. 1992. *A CAD Interpreter for Prismatic Components* Proceedings of the Eighth National Conference for Manufacturing Research, Sep.

Mortensen K.S and Belnap B.K. 1989. *A Rule-Based Approach Employing Feature Recognition for Engineering Graphics Characterisation* Computer Aided Engineering Journal, Dec.

Pratt M.J 1988. *Synthesis of an Optimal Approach to Form Feature Modelling* Computers In Engineering.

Shah J.J and Rogers M.T. 1988. *Functional Requirements and Conceptual Design of the Feature-Based Modelling System* Computer Aided Engineering Journal, Feb.

Vosniakos G.C. and Davies B.J 1990. *An IGES Post-Processor for Interfacing CAD and CAPP of 2-1/2D Prismatic Parts* International Journal of Advanced Manufacturing Technology 5, pp. 135-164.

Wang H.P. and Wysk R.A 1987. *Intelligent Reasoning for Process Planning* Computer In Industry 8, pp. 293-309.

Wang H.P. and Wysk R.A 1988. *AIMSI: A Preclude to a New Generation of Integrated CADCAM Systems* Int. J. Prod. Res, Vol 26, No 1.

Zhang Y. and Mileham A.R 1990. *A CAD Interpreter for Interfacing CAD and CAPP of 2D Rotational Parts* Proceedings of the Sixth International Conference on CAD/CAM, Robotics etc, pp.333-336.

A STRATEGY FOR EXTRACTING MANUFACTURING FEATURES FOR PRISMATIC COMPONENTS FROM CAD

Mr S. Linardakis and Dr A.R. Mileham

School of Mechanical Engineering, University of Bath
Claverton Down, BATH BA2 7AY

A novel approach for interfacing CAD and CAPP for prismatic components is discussed. The analysis presented forms part of ongoing research on a CAD interpreter using the industry standard DXF file format to extract and classify process planning information from a CAD engineering drawing. The DXF file of a typical 3 view engineering drawing of a prismatic component is used as the system input. This is then processed in order to identify a certain range of manufacturing features. The methodology for identifying features is presented, along with a typical application example of how this has been implemented within the system.

Introduction

Recently the concept of using component features for design and manufacturing applications has received much attention and research effort. Describing components by the use of features is seen by many as the key to genuine integration of the many aspects of design and planning of manufacture, particularly in a modern computer-controlled environment incorporating Computer Aided Design (CAD) and Computer Aided Process Planning (CAPP) (Brimson and Downey 1986). Features originate in the reasoning processes used in various design, analysis, and manufacturing activities (Cunningham and Dixon 1988) and are frequently strongly associated with particular application domains. Hence there are many different definitions of features. A broad definition in the engineering domain is given by (Chang 1990) as: " a subset of geometry on an engineering part which has a special design or manufacturing characteristic".

Two approaches to using features in manufacturing applications have been used by researchers: Feature Identification and Feature Based Design. Feature Identification (Choi et al. 1984) is the process of extracting manufacturing features from a CAD database. In Feature Based Design (Shah and Rogers 1988) the designer uses manufacturing features to define an engineering part and build the

CAD database. While this provides a natural transition from design to manufacturing, it also imposes limitations on the generality and extensibility of features, by enforcing an interdependence between design and manufacturing capabilities (Chang 1990).

From the design point of view, one of the attractions of geometric modelling within CAD is that a well constructed modeller is capable of representing all the geometric aspects of a component within a chosen domain. If a complete geometric description is available, then it is clearly possible to use computer methods to interrogate this information and transform it into any desired form. Thus collections of surfaces could be recognised as features and transformed into manufacturing features for CAPP purposes. Much research work has been done in this Feature Identification field.

Mortensen and Belnap (1989), describe a methodology employing feature recognition for rotational parts from a 2D CAD system database (although an interactive graphics system was used to generate the test data). However this system is limited as it lacked automatic dimension extraction. An IGES (Initial Graphics Exchange Specification) file format post processor has been reported (Vosniakos and Davies 1990), which can extract the data that defines a 2-1/2D prismatic component, based on a wire-frame IGES file. However, the awkward points of the IGES standard makes it impossible to transfer the full context of a product model, unless a series of restrictions are applied. An identifier able to extract geometric information from a rotational part data file (generated using a 2D wire-frame CAD modeller) has also been developed (Wang and Wysk 1987, 1988). However, information such as tolerance, surface roughness etc is still typed interactively. The STEP protocol is being formulated by the International Standards Organisation to be the future standard in data transfer. However, since it is still being drafted, it is not yet widely accepted by industry, and incompatible with the current generation of commercial CAD systems (Sivayoganathan, Balendran, Czerwinski, Keats, Leibar, and Seiler 1993). Zhang and Mileham (1990) developed an interpreter using the DXF format for 2D rotational parts that can extract tolerance information automatically. Although all these approaches have contributed significantly to this domain, research on the topic is far from being conclusive.

Interpreting CAD Data for Prismatic Components

The work presented in this paper is part of ongoing research on a CAD Interpreter for Prismatic Components being developed at Bath University (Linardakis and Mileham 1992) aimed at minimising human intervention on the interface of CAD and CAPP. The interpreter identifies the geometric and tolerance data that define a prismatic component from a wireframe-based CAD DXF file. The DXF file format is a current industry standard, enabling data exchange between different CAD packages, hence by using it as an input, the interpreter retains compatibility. An algorithm has already been developed to extract and classify the geometric entities (lines, circles, arcs etc) of a drawing.

For a prismatic part, complete manufacturing information can be provided by a three-view drawing in 2D space using orthographic projection. However, the duplication of the same features on different views of the drawing could confuse

the interpretation process. This is avoided by placing the different views of the drawing in different layers. The views are layered in a predefined convention. Since the layer that an entity belongs to is also reflected in its entry in the DXF file, the profile recognition and feature identification task becomes easier.

An algorithm has been developed to identify the external profile of a component (Linardakis and Mileham 1993). This enables the interpreter to determine the shape envelope dimensions and hence deduce the minimum shape envelope of the raw material required for the component to be machined. The objective of the research work is to then convert the CAD geometric entity information into a manufacturing feature based model, which can be used to drive a process planning system directly.

A Methodology for Extracting Manufacturing Features

An algorithm is under development to identify and extract certain external profile manufacturing features from the sorted data file of the CAD Interpreter. In order to locate and classify a feature, the Feature Extractor proceeds in the following manner:

(1) The extracted line entities and their coordinates from the FRONT view are first examined. The edges of the component are formed by the lines with the minimum and maximum X and Y coordinates :

Line (Xmin, Ymin) to (Xmin, Ymax) forms an edge.
Line (Xmin, Ymax) to (Xmax, Ymax) forms an edge.
Line (Xmax, Ymax) to (Xmax, Ymin) forms an edge.
Line (Xmax, Ymin) to (Xmin, Ymin) forms an edge.

(2) The identified edges are next checked for continuity in a clockwise direction. If there is a continuous line joining two maxima-minima points, then there is no feature on that particular edge. If however, an edge-line is interrupted, then a feature exists on that edge, and the Feature Identification program is called. This consists of several recognition algorithms for different features (currently under development), which are consecutively applied until a particular feature is identified and extracted. The next edge is then examined, and so on, until all the edges have been processed.

(3) The same method is applied to the extracted edges from the PLAN and END views of the component.

A sample component showing the methodology employed can be seen in Figure 1. In this typical application example, a through slot needs to be machined on the part. The through slot recognition algorithm is applicable here, a description of which is given in the next section.

The Through Slot Recognition Algorithm

Consider the component shown in Figure 1, on which a plain through slot needs to be machined. This can be seen on the FRONT and PLAN views of the drawing.

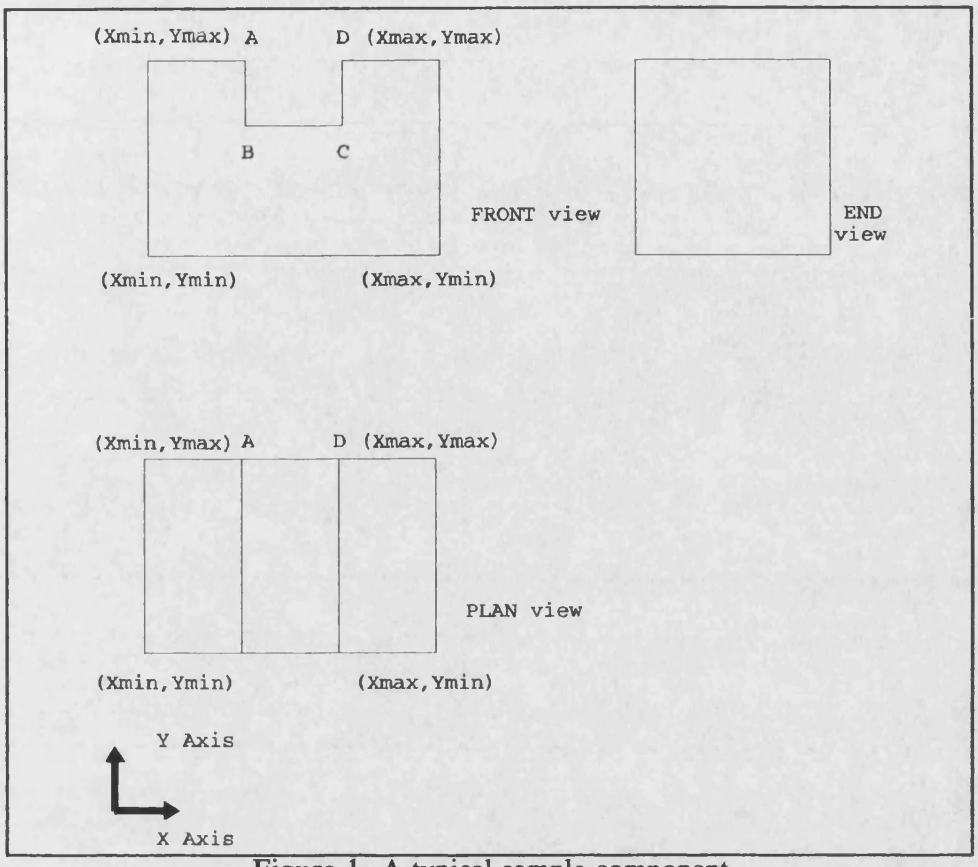


Figure 1. A typical sample component

In order to verify that the feature encountered is indeed a plain through slot, the Feature Extractor proceeds in the following manner:

(1) Point A is located (Xmin < X < Xmax, Ymax) where the discontinuity starts.
(2) Point B is found, and line AB confirmed as a continuous line (no secondary features) vertical to line (Xmin, Ymax)A.
(3) Point C is located, and line BC confirmed as a continuous line vertical to AB. Line BC should also be parallel to edge (Xmin, Ymax)(Xmax, Ymax).
(4) Point D is found, and line CD confirmed as a continuous line vertical to BC and parallel to AB.
(5) Line D(Xmax, Ymax) is found and its continuity confirmed.
(6) The PLAN view is examined. The Extractor checks to see if there are two lines starting from points A and D and finishing at Ymin, parallel to each other and to the two edges. If there are, a through slot is confirmed and extracted.

The above algorithm has been adapted to also detect a blind slot.

Conclusions

A novel approach for extracting manufacturing features for prismatic components from CAD DXF files is being developed. The Feature Extractor presented uses a sorted data file from a CAD DXF Interpreter to locate and classify a range of manufacturing features. This is achieved by consecutively applying various feature recognition algorithms (of which the through slot is discussed here as a typical example) on the component's external profile. The system is being validated over a range of prismatic components and represents a further step towards CAD/CAPP integration.

References

Brimson J.A. and Downey P.J. 1986. Feature Technology: A Key to Manufacturing Integration *CIM Review*, Spring.

Cunningham J.J and Dixon J.R. 1988. Designing with Features: The Origin of Features *Computers in Engineering*, 1.

Chang T.C. 1990. *Expert Process Planning for Manufacturing* (Addison-Wesley Publishing Company).

Choi B., Barash M.M. and Anderson D.C. 1984. Automatic Recognition of Machined Surfaces from a 3D Solid Model *Computer Aided Design*, 16, 81-86.

Linardakis S. and Mileham A.R. 1992. A CAD Interpreter for Prismatic Components *Proceedings of the Eighth National Conference for Manufacturing Research*, 21-25.

Linardakis S. and Mileham A.R. 1993. Manufacturing Feature Identification for Prismatic Components from CAD DXF Files *Proceedings of the Ninth National Conference for Manufacturing Research*, 37-41.

Mortensen K.S and Belnap B.K. 1989. A Rule-Based Approach Employing Feature Recognition for Engineering Graphics Characterisation *Computer Aided Engineering Journal*, Dec.

Pratt M.J 1988. Synthesis of an Optimal Approach to Form Feature Modelling *Computers In Engineering*.

Shah J.J and Rogers M.T. 1988. Functional Requirements and Conceptual Design of the Feature-Based Modelling System *Computer Aided Engineering Journal*, Feb.

Sivayoganathan K., Balendran V., Czerwinski A., Keats J., Leibar A.M. and Seilar A. 1993. CAD/CAM Data Exchange Application, *Proceedings of the Ninth National Conference for Manufacturing Research*, 306-310.

Vosniakos G.C. and Davies B.J 1990. An IGES Post-Processor for Interfacing CAD and CAPP of 2-1/2D Prismatic Parts *International Journal of Advanced Manufacturing Technology*, 5, 135-164.

Wang H.P. and Wysk R.A 1987. Intelligent Reasoning for Process Planning *Computers In Industry*, 8, 293-309.

Wang H.P. and Wysk R.A 1988. AIMSI: A Preclude to a New Generation of Integrated CADCAM Systems *International Journal of Production Research*, 26.

Zhang Y. and Mileham A.R 1990. A CAD Interpreter for Interfacing CAD and CAPP of 2D Rotational Parts *Proceedings of the Sixth International Conference on CAD/CAM, Robotics etc*, 333-336.

A METHODOLOGY FOR IDENTIFYING FEATURES FOR PRISMATIC COMPONENTS FROM CAD FILES

Mr S. Linardakis and Dr A.R. Mileham

School of Mechanical Engineering, University of Bath
Claverton Down, BATH BA2 7AY

A novel approach for interfacing Computer Aided Design (CAD) and Computer Aided Process Planning (CAPP) for prismatic components is discussed. The analysis presented forms part of research work on a CAD Interpreter and Feature Processor, which uses the industry standard DXF file format to translate CAD drawings into a manufacturing feature based output file, that can be directly used by a CAPP system. The DXF file of a typical 3 view engineering drawing of a prismatic component is used as the system input. This is then processed in order to identify a certain range of manufacturing features. The strategy for extracting multiple secondary features (embedded within other features), in particular through slots and steps are discussed, along with a typical application example.

Introduction

The use of features is a prominent trend in recent years in research into the application of Computer Aided Design (CAD) / Computer Aided Manufacturing (CAM) systems. Feature technology is viewed by many as the key to genuine integration of the many aspects of design and planning of manufacture, particularly in a modern computer-controlled environment, incorporating CAD and Computer Aided Process Planning (CAPP) (Ajmal and Zhang 1994).

Two approaches to using features in manufacturing applications have been used by researchers: Feature Based Design and Feature Identification. In Feature Based Design (Case and Gao 1993) the designer uses manufacturing features to define an engineering part and hence build the CAD database. While this provides a natural transition from design to manufacturing, it also imposes limitations on the generality and extensibility of features (feature libraries tend to be limited, and feature operations such as add, delete, edit etc are sometimes insufficient) (Chang 1990).

Feature Identification (Chuang and Henderson 1990) is the process of extracting manufacturing features from a conventional CAD system, such as a two-dimensional drafting system, wireframe modeller or solid modeller. This usually involves complicated reasoning logic and data processing. To overcome these complexities, the STEP protocol is being formulated by the International Standards Organisation to be the future standard in data transfer. However, since it is still being drafted, it is not yet widely used by industry, and incompatible with the current generation of commercial CAD systems (Sivayoganathan, Balendran, Czerwinski, Keats, Leibar, and Seiler 1993).

Most of the current research on Feature Identification is based on 3-D solid models (Case and Gao 1993). Due to the inadequacy of geometrical and topological information in representing a 3-D object, wireframe models have undergone little similar research work on Feature Recognition. However, wireframe modellers are the first generation of geometric modelling systems and are widely used in industry (Zhao, Blount, Jones and Baines 1993). Most commercially available CAD packages have been developed on wireframe modelling systems. Within these systems, the components are represented with lower level entities, such as lines, circles and arcs. In manufacturing, components are usually completely described using drawings. These types of descriptions are closely related to wireframe models and are the most widely used information media through different stages of manufacturing. Therefore, extractions of features of manufacturing significance based on wireframe models is necessary, especially for machining (Zhao and Blount 1990).

Extracting Manufacturing Features for Prismatic Components from CAD

The work presented in this paper is part of ongoing research on a CAD Interpreter and Feature Processor for Prismatic Components being developed at Bath University (Linardakis and Mileham 1992) aimed at minimising human intervention on the interface of CAD and CAPP. The interpreter identifies the geometric and tolerance data that defines a prismatic component from a wireframe-based CAD DXF file. The DXF file format is a current industry standard, enabling data exchange between different CAD packages, hence by using it as an input, the interpreter retains compatibility. An algorithm has already been developed to extract and classify the geometric entities (lines, circles, arcs etc) of a drawing (Linardakis and Mileham 1992).

For a prismatic part, complete manufacturing information can be provided by a three-view drawing in 2-D space using orthographic projection. However, the duplication of the same features on different views of the drawing could confuse the interpretation process. This is avoided by placing the different views of the drawing in different layers. The views are layered in a predefined convention. Since the layer that an entity belongs to is also reflected in its entry in the DXF file, the profile recognition and feature identification task becomes easier.

An algorithm has been developed to identify the external profile of a component (Linardakis and Mileham 1993). This enables the interpreter to determine the shape envelope dimensions and hence deduce the minimum shape envelope of the raw material required for the component to be machined. Algorithms for identifying simple through slots, simple steps and plain holes have

also been developed. The general strategy for the process, as well as the simple through slot recognition algorithm can be found in (Linardakis and Mileham 1994). The work presented in this paper deals with the extraction of multiple secondary features (embedded within other features), in particular multiple embedded through slots, steps, or any combination of the two.

Identifying Multiple Embedded Through Slots

To assist in visualising the methodology employed by the Feature Processor, consider the sample component shown in Figure 1. In this typical application example, multiple through slots and steps need to be machined on the part. These can be seen on the front and plan views of the drawing. In order to locate and classify the through slots, the Feature Processor proceeds in the following manner:

(1) The lines and their coordinates for every view of the drawing are read in from the interpreted DXF file. They are then sorted in a "serial" way (that is the end of a line forms the beginning of the next one). At this point, and to facilitate later checking, a new coordinate system is implemented (compared to the one used in the DXF file), whereas the point of origin for every view of the drawing is defined as the bottom left hand corner of each view. Line coordinates are changed to reflect this modification.
(2) The extracted line entities and their coordinates from the front view are first examined. The Feature Processor starts "walking" around the edges of the component (starting from the top left hand side, ie Corner 0, Edge 0 in a clockwise direction), looking for discontinuities which indicates the presence of features. In this example, a discontinuity is found at Edge 0.
(3) The Feature Extractor then selects the first "downwards" heading line (from the ordered array of lines) and flags it as the "reference" line. It then continues "walking" around the ordered lines looking for one or more "upwards" lines whose Y coordinates (in this case) correspond to the reference line. These indicate the other side of the through slot(s). Once these lines are found, a through slot(s) is identified with a depth equal to the lengths of the upwards line(s) with regard to the reference line. The width of the slot(s) is the distance between the reference line and the upwards line(s). In the sample component, slot A is thus identified first.
(4) The Feature Extractor then proceeds with the next downwards heading line which flags as the new "reference" line. It then applies the above procedure to find the next slot(s), and so on until all the downwards lines for Edge 0 are examined. In this example, the system thus picks up slots B and C.
(5) The program continues with the next edge (Edge 1) of the front view of the component, applying the same process but in a modified form (this time looking for reference lines heading right-to-left and their opposites). Edges 2 and 3 are next examined, again with modified forms of the algorithm, until the program finishes at the point where it started (ie Corner 0, Edge 0). The above methodology is next applied to the plan and then the end views of the drawing respectively.

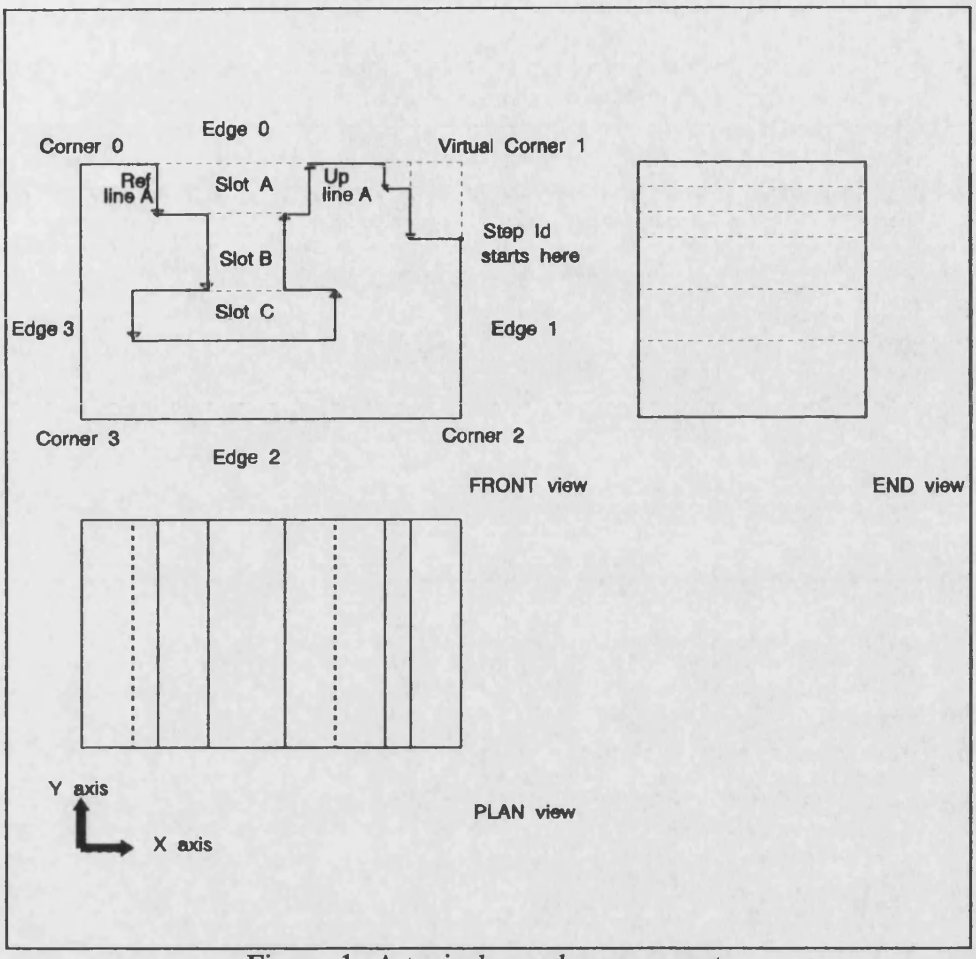


Figure 1. A typical sample component.

Identifying Multiple Steps

A step can be found at any corner of the drawing of a component which does not have an edge line starting from or ending at it (ie if the corner is not real but "virtual"). In the example shown in Figure 1, Corner 1 of the front view is such a corner, containing two steps. In order to locate and classify such steps, the Feature Processor proceeds as follows:

(1) Starting from Corner 0 of the front view, the Feature Processor "walks" around the component looking for virtual corners. In this example, Corner 1 is the first (and only) such corner it comes across.
(2) Once a virtual corner is found (indicating a step), the program flags the closest real point to the corner of the next edge (as can be seen in Figure 1) and then traces back in an anti-clockwise direction until it finds the closest real point to the corner of the current edge.
(3) A step (or steps) is formed by the number of "downwards" facing lines found between these real points. Every downwards line forms a further step. The width of each step is the distance between the starting real point and the downwards line (or in the case of multiple steps the distance between subsequent downwards lines).

The depth of the step is the length of the downwards line. In the sample component, two such lines exist between the two real points closest to the virtual corner, indicating two steps.
(4) The same algorithm (in a slightly modified form depending on where the virtual corner is) is applied for any other virtual corner of the front view, and then the plan and end views of the drawing respectively.

Conclusions

A novel approach for extracting manufacturing features for prismatic components from CAD DXF files is being developed. The Feature Processor presented uses a sorted data file from a CAD DXF Interpreter to locate and classify multiple secondary features (embedded within other features). This is achieved by consecutively applying various feature recognition algorithms (of which the through slots and steps are discussed here) on the component's external profile. The system is being validated over a range of prismatic components and has proved very robust, representing a further step towards CAD/CAPP integration.

References

Ajmal A. and Zhang H.G. 1994. State of the Art Review of CAPP: Its Impact and its Application in CIM *Proceedings of the Tenth National Conference on Manufacturing Research*, 239-243.

Case K. and Gao X. 1993. *Feature Technology - An Overview* Int J of Computer Integrated Manufacture, Vol. 6, Nos 1&2, 2-12.

Chang T.C. 1990. *Expert Process Planning for Manufacturing* (Addison-Wesley Publishing Company).

Chuang S.H. and Henderson M.R. 1990. *Three - Dimensional Shape Pattern Recognition and Vertex Classification and Vertex Edge Graphs* Computer Aided Design Vol. 22, No 6, July/August.

Linardakis S. and Mileham A.R. 1992. A CAD Interpreter for Prismatic Components *Proceedings of the Eighth National Conference on Manufacturing Research*, 21-25.

Linardakis S. and Mileham A.R. 1993. Manufacturing Feature Identification for Prismatic Components from CAD DXF Files *Proceedings of the Ninth National Conference on Manufacturing Research*, 37-41.

Linardakis S. and Mileham A.R. 1994. A Strategy for Extracting Manufacturing Features for Prismatic Components from CAD *Proceedings of the Tenth National Conference on Manufacturing Research*, 299-303.

Sivayoganathan K., Balendran V., Czerwinski A., Keats J., Leibar A.M. and Seilar A. 1993. CAD/CAM Data Exchange Application, *Proceedings of the Ninth National Conference for Manufacturing Research*, 306-310.

Zhao Z., Blount G.N., Jones R.M. and Baines R.W. 1993. Representation of Surfaces in 3-D Wireframe Models *Proceedings of the Thirtieth International Matador Conference*, 573-579.

Zhao Z. and Blount G.N. 1990. Recognition of Machined Surfaces for Manufacturing Based on Wireframe Models *Proceedings of the International Conference on Developments in Forming Technology*, Portugal, 362-368.

A METHODOLOGY FOR EXTRACTING MANUFACTURING FEATURES FOR PRISMATIC COMPONENTS FROM CAD DXF FILES

S. Linardakis and A.R. Mileham

School of Mechanical Engineering, University of Bath, Claverton Down,
BATH BA2 7AY, U.K.

ABSTRACT

A novel approach for interfacing Computer Aided Design (CAD) and Computer Aided Process Planning (CAPP) for prismatic components is discussed. The analysis presented forms part of research work on a CAD interpreter and feature extractor, which uses the industry standard DXF (Drawing Interchange File) file format to translate CAD drawings into a manufacturing feature based output file, to be directly used by a CAPP system. The DXF file of a typical 3 view engineering drawing of a prismatic component is used as the system input. This is then processed in order to identify a certain range of manufacturing features. The strategy for identifying cylindrical features (holes and threads) as well as pockets is discussed, along with a typical example of how this has been implemented.

1. INTRODUCTION

The use of features is a prominent trend in recent years in research into the application of Computer Aided Design (CAD) / Computer Aided Manufacturing (CAM) systems. Feature technology is viewed by many as the key to genuine integration of the many aspects of design and planning of manufacture, particularly in a modern computer-controlled environment, incorporating CAD and Computer Aided Process Planning (CAPP) [1].

Two approaches to using features in manufacturing applications have been used by researchers: Feature Based Design and Feature Identification. In Feature Based Design [2], the designer uses manufacturing features to define an engineering part and hence build the CAD database. While this provides a natural transition from design to manufacturing, it also imposes limitations on the generality and extensibility of features (feature libraries tend to be limited, and feature operations such as add, delete, edit etc are sometimes insufficient) [3].

Feature Identification [4] is the process of extracting manufacturing features from a conventional CAD system, such as a two-dimensional drafting system, wireframe modeller or solid modeller. This usually involves complicated reasoning logic and data processing. To overcome these complexities, the STEP protocol is being formulated by the International Standards Organisation to be the future standard in data transfer. However, since it is still being drafted, it is not yet widely used by industry, and incompatible with the current generation of commercial CAD systems [5].

Most of the current research on Feature Identification is based on 3-D solid models [2]. Due to the inadequacy of geometrical and topological information in representing a 3-D object, wireframe models have undergone little similar research work on Feature Recognition. However, wireframe modellers are the first generation of geometric modelling systems and are widely used in industry [6]. Most commercially available CAD packages have been developed on wireframe modelling systems. Within these systems, the components are represented with lower level entities, such as lines, circles and arcs. In manufacturing, components are usually completely described using drawings. These types of descriptions are closely related to wireframe models and are the most widely used information media through different stages of manufacturing. Therefore, extractions of features of manufacturing significance based on wireframe models is necessary, especially for machining [7].

2. EXTRACTING MANUFACTURING FEATURES FOR PRISMATIC COMPONENTS FROM CAD

The work presented in this paper is part of ongoing research on a CAD Interpreter and Feature Processor for Prismatic Components being developed at Bath University [8] aimed at minimising human intervention on the interface of CAD and CAPP. The interpreter identifies the geometric and tolerance data that defines a prismatic component from a wireframe-based CAD DXF file. The DXF file format is a current industry standard, enabling data exchange between different CAD packages, hence by using it as an input, the interpreter retains compatibility. An algorithm has already been developed to extract and classify the geometric entities (lines, circles, arcs etc) of a drawing [8].

For a prismatic part, complete manufacturing information can be provided by a three-view drawing in 2-D space using orthographic projection. However, the duplication of the same features on different views of the drawing could confuse the interpretation process. This is avoided by placing the different views of the drawing in different layers. The views are layered in a predefined convention. Since the layer that an entity belongs to is also reflected in its entry in the DXF file, the profile recognition and feature identification task becomes easier.

An algorithm has been developed to identify the external profile of a component [9]. This enables the interpreter to determine the shape envelope dimensions and hence deduce the minimum shape envelope of the raw material required for the component to be machined. Algorithms for identifying multiple through slots and multiple steps have also been developed. The general strategy for the process, as well as the simple through slot recognition algorithm can be found in Linardakis and Mileham [10]. The work presented in this current paper deals with the extraction of cylindrical features (holes and threads) as well as pockets.

3. IDENTIFYING CYLINDRICAL FEATURES

To assist in visualising the methodology employed by the Feature Processor, consider the sample component shown in Fig. 1. In this typical application example, a stepped, conical-ended counterbored hole and a close pocket need to be machined on the part. These can be seen on the front, plan and end views of the drawing. In order to locate and classify the hole, the Feature Processor proceeds in the following manner:

(1) The entities (lines, circles, arcs) and their coordinates for every view of the drawing are read in from the interpreted DXF file. They are then sorted in a "serial" way (that is the end of a line forms the beginning of the next one). At this point, and to facilitate later checking, a new coordinate system is implemented (compared to the one used in the DXF file), whereas the point of origin for every view of the drawing is defined as the bottom left hand corner of each view. Coordinates for all entities are changed to reflect this modification.
(2) The extracted circle entities and their coordinates from the front view are first examined. A circle indicates the presence of a hole, while two concentric circles indicate a stepped hole. The Feature Processor can also pick up threads; these are formed by a concentric circle and a 270 degree arc. In this example, there are two concentric circles on the front view (ie a stepped hole).
(3) The Feature Extractor then selects the first detected hole and its coordinates. It determines the hole's type (plain, stepped or threaded), position, if it is "normal" or "hidden" (if originating from the directly opposite side of the component), and its diameter(s). The latter is obtained from the interpreted DXF file. In the sample component shown, a stepped hole is deduced, and its diameters reported.
(4) The program next correlates the hole's diameter coordinates to the plan view of the drawing, and examines the line entities found in these coordinates. It thus deduces the hole(s)' depth(s), using a developed algorithm. The system is also able to distinguish and report if a stepped hole is counterbored or countersunk, as well as if the hole is through, flat-ended or conical-ended. In the example in Fig. 1, a counterbored conical-ended hole is hence recognised, and its depth reported.
(5) The Feature Extractor continues with the next hole, applying the same process to determine its parameters until all holes in the front view are found. The above methodology is next applied to the plan and then the end views of the drawing respectively, but in a slightly modified form (this time relaying hole coordinates to the front view to determine the depths).

4. IDENTIFYING POCKETS

A pocket can be found at any corner of the drawing of a component ("open" pocket), along any of its edges ("side" pocket) or within its internal volume ("closed" pocket), as shown on Fig. 2. For the purposes of this research only orthogonal pockets are considered (pockets containing 90 or 180 degree arcs), since they cover the majority of typical applications. In the example shown in Fig. 1, a closed pocket consisting of two 180 degree arcs interconnected with lines ("extension" lines) can be seen on the front view of the drawing. In order to locate and classify such a pocket, the Feature Processor proceeds as follows:

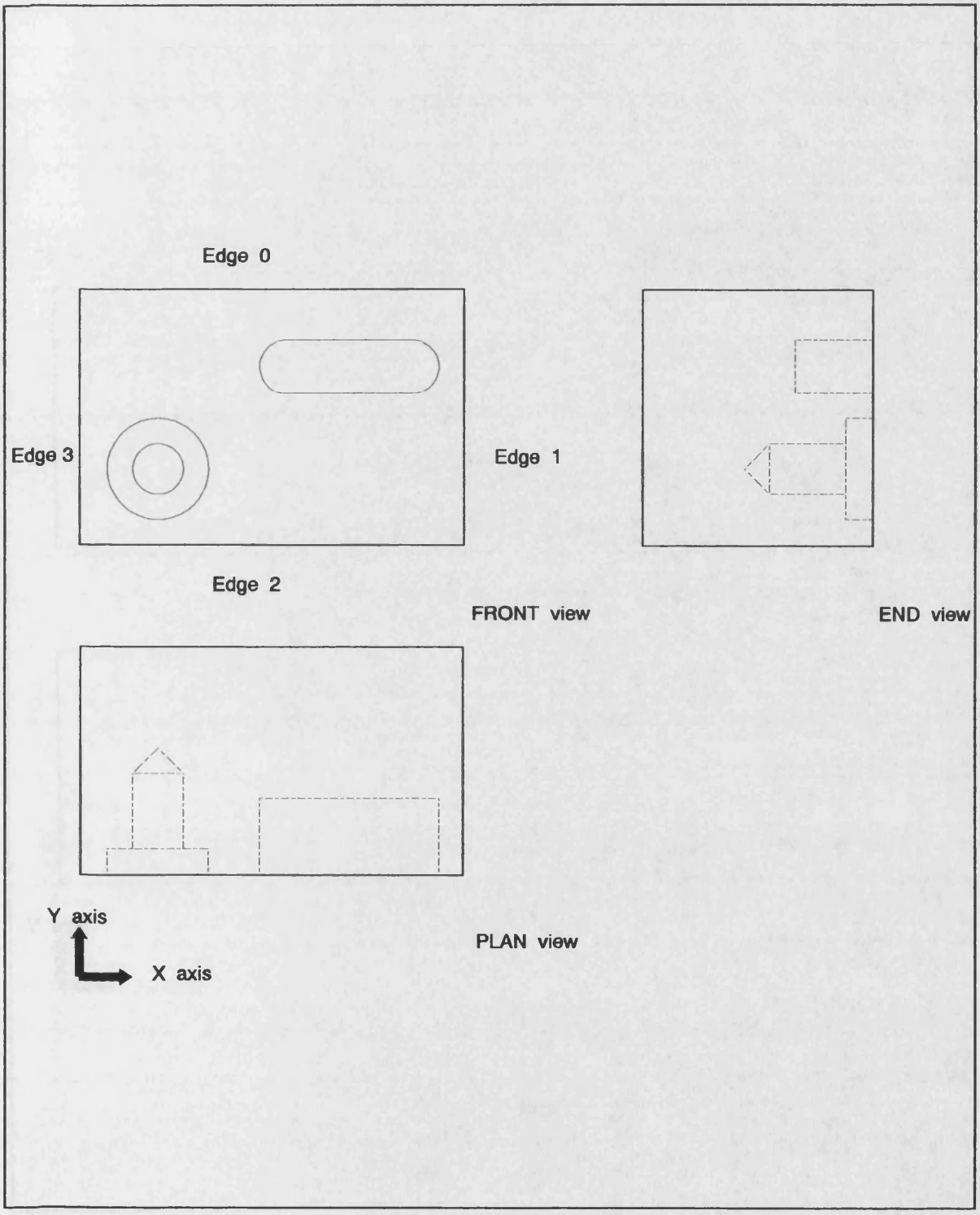


Figure 1. A typical sample component.

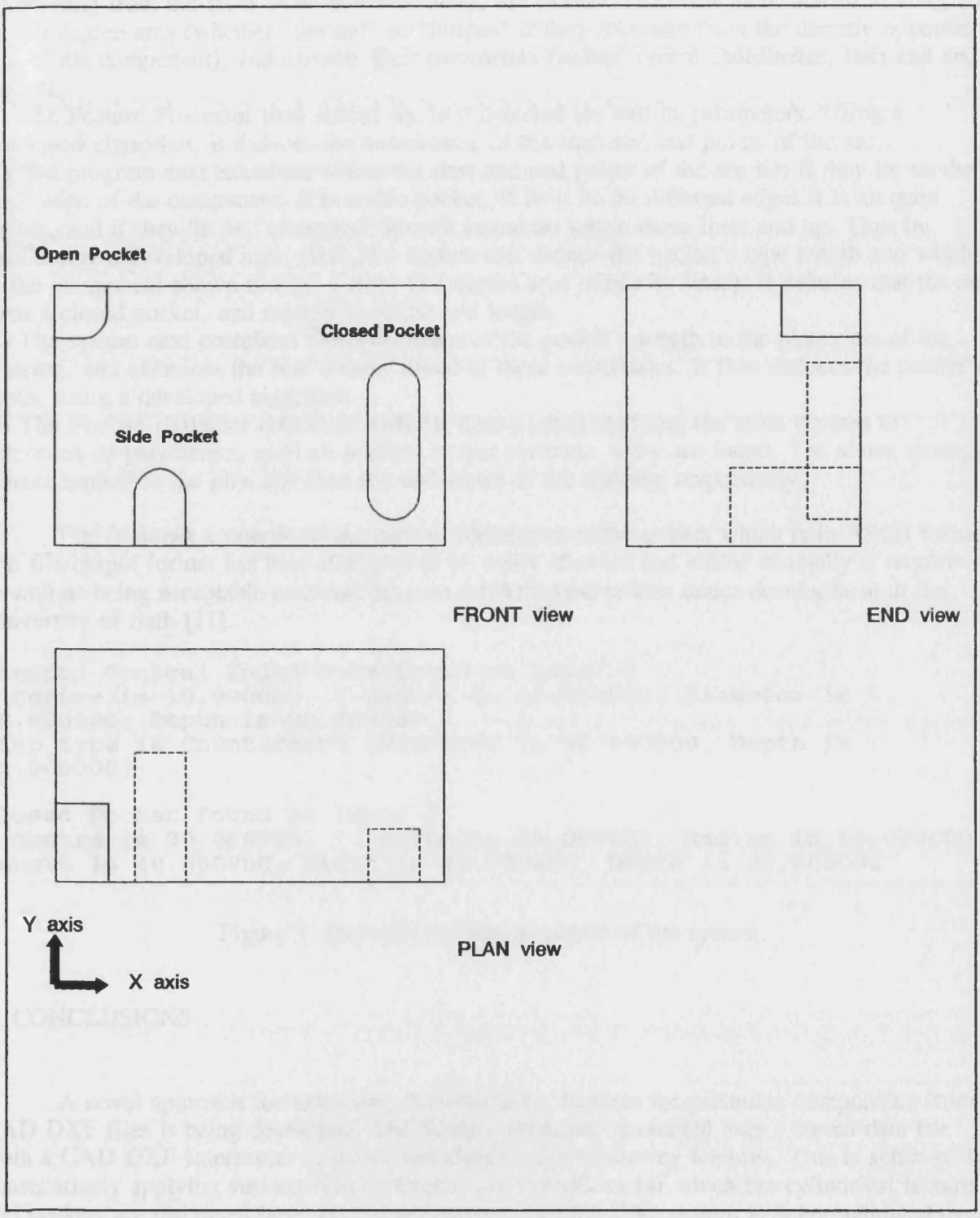


Figure 2. Types of pockets identified by the system.

(1) Starting from the front view of the drawing, the Feature Processor identifies all existing 90 or 180 degree arcs (whether "normal" or "hidden" if they originate from the directly opposite side of the component), and extracts their parameters (radius, centre coordinates, start and end angles).
(2) The Feature Processor then selects the first detected arc and its parameters. Using a developed algorithm, it deduces the coordinates of the start and end points of the arc.
(3) The program next examines where the start and end points of the arc lie: if they lie on the same edge of the component, it is a side pocket, if they lie on different edges it is an open pocket, and if they lie on "extension" lines it examines where these lines end up. Thus by questioning a developed logic table, the system can deduce the pocket's type length and width. In the component shown in Fig. 1 (two 180 degree arcs joined by lines), it deduces that the arcs form a closed pocket, and reports its width and length.
(4) The system next correlates the coordinates of the pocket's length to the plan view of the drawing, and examines the line entities found in these coordinates. It thus deduces the pocket's depth, using a developed algorithm.
(5) The Feature Extractor continues with the next pocket, applying the same process to determine its parameters, until all pockets in that particular view are found. The above strategy is next applied to the plan and then the end views of the drawing respectively.

Fig. 3 shows a sample of the current file output of the system which is in ASCII format. The file output format has been designed to be easily checked and edited manually if required, as well as being acceptable as direct input to a CAPP system also under development at the University of Bath [11].

```
Stepped Conical Ended Hole found on layer 0
X Centre is 10.000000, Y Centre is 20.000000, Diameter is
10.000000, Depth is 20.000000
Step type is Counterbore (Diameter is 20.000000, Depth is
10.000000)

Closed Pocket found on layer 0
X Centre is 30.000000, Y Centre is 60.000000, Radius is 10.000000
Length is 40.000000, Width is 20.000000, Depth is 30.000000
```

Figure 3. Example file format output of the system.

5. CONCLUSIONS

A novel approach for extracting manufacturing features for prismatic components from CAD DXF files is being developed. The Feature Processor presented uses a sorted data file from a CAD DXF Interpreter to locate and classify manufacturing features. This is achieved by consecutively applying various feature recognition algorithms (of which the cylindrical features and pockets are discussed here) on the component's profile. The system is being validated over a range of prismatic components and has proved very robust, representing a further step towards CAD/CAPP integration.

REFERENCES

1. Ajmal A. and Zhang H.G., State of the Art Review of CAPP: Its Impact and its Application in CIM, Proceedings of the Tenth National Conference on Manufacturing Research, (1994), pp239-243. 2. Case K. and Gao X., Feature Technology - An Overview, Int J of Computer Integrated Manufacture, Vol. 6, Nos 1&2, (1993), 2-12.
3. Chang T.C., Expert Process Planning for Manufacturing, Addison-Wesley Publishing Company, USA, (1990).
4. Chuang S.H. and Henderson M.R., Three - Dimensional Shape Pattern Recognition and Vertex Classification and Vertex Edge Graphs, Computer Aided Design, Vol. 22, No 6, (1990), July/August.
5. Sivayoganathan K., Balendran V., Czerwinski A., Keats J., Leibar A.M. and Seilar A., CAD/CAM Data Exchange Application, Proceedings of the Ninth National Conference for Manufacturing Research, (1993), pp306-310.
6. Zhao Z., Blount G.N., Jones R.M. and Baines R.W., Representation of Surfaces in 3-D Wireframe Models, Proceedings of the Thirtieth International Matador Conference, (1993), pp573-579.
7. Zhao Z. and Blount G.N., Recognition of Machined Surfaces for Manufacturing Based on Wireframe Models, Proceedings of the International Conference on Developments in Forming Technology, Portugal, (1990), pp362-368.
8. Linardakis S. and Mileham A.R., A CAD Interpreter for Prismatic Components, Proceedings of the Eighth National Conference on Manufacturing Research, (1992), pp21-25.
9. Linardakis S. and Mileham A.R., Manufacturing Feature Identification for Prismatic Components from CAD DXF Files, Proceedings of the Ninth National Conference on Manufacturing Research, (1993), pp37-41.
10. Linardakis S. and Mileham A.R., A Strategy for Extracting Manufacturing Features for Prismatic Components from CAD, Proceedings of the Tenth National Conference on Manufacturing Research, (1994), pp299-303.
11. Rustom E., Zhang Y.F., Isik I. and Mileham A.R., The Development of a Generative Process Planning System, Advanced Manufacturing Seminar Series, Coventry (1988).